Adam Mickiewicz University in Poznań

Andrzej Gajda

Abductive hypotheses generation in neural-symbolic systems

A dissertation submitted
in partial fulfillment of the requirements
for the degree of Doctor of Philosophy
in
Cognitive Science

Supervisor:
Mariusz Urbański, PhD, Dr. Habil.

Department of Logic and Cognitive Science
Institute of Psychology

June 2, 2018

For my beloved wife

Acknowledgements

I would like to thank professor Mariusz Urbański for constant support and understanding the challenges of pursuing multidirectional scientific career. I am greatly inspired by His professional abitiousness and educative approach.

I also acknowledge scientific collaboration with Adam Kupś on the implementation of the abductive procedure.

I am much obliged to all my colleagues from the Reasoning Research Group—it was great to share the laboratory with you during the last couple of years.

My gratitude is also due to my wife upholding me and my spirit throughout the entire process.

Contents

Chapter 1
Introduction

The general form of abductive reasoning can be described by the following schema proposed by Peirce [40, 5.189]: from the observed phenomenon A and from the known rule *if* H *then* A, infer H. This schema can be interpreted in many ways and one of them is that abductive reasoning can be used to *fill the gap* between a knowledge base Γ and a phenomenon ϕ, which is unattainable from Γ itself [58]. According to an algorithmic point of view advocated by Gabbay and Woods an abductive hypothesis (or abducible) 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" [12, p. 28], where ϕ, a formula to be proven or computed, is an abductive goal. Let us designate by Γ' the knowledge base Γ augmented with an abducible H. From a slightly more general point of view than the one offered by Gabbay and Woods we may claim that an abductive hypothesis is the difference between Γ' and Γ. The term 'difference' can be understood here as a symmetric difference. Γ' may result from Γ not only by just *addition* of new information but also by *removal* from or *modification* of some information in Γ (some similarity can be found in *contraction* and *revision* operations in belief revision theory, see e.g. [19]). This is the way in which the concept of an abductive hypothesis will be understood in this work.

My aim is to define such abductive procedure that is capable of generation of abductive hypotheses as a symmetric difference between initial and final knowledge base. On the one hand, in order to be able to strictly define what it means to compare two different knowledge bases they ought to be formalised. On the other hand, there is no efficient automated formal system that is able to revise given knowledge base in a way that the information which is unattainable from such a knowledge base becomes attainable. However, there are systems in the domain of machine learning algorithms and artificial neural networks that can be trained with "real world examples" in order to enhance their ability to model phenomenons they were designed to model. In particular, when they are trained in a suitable way, they can obtain "new

knowledge" and what follows, represent information that they were not able to represent before the training process. Therefore, the main goal of this work can be further specified as a design of such integrated system that includes the following properties:

- knowledge base and abductive goal can be represented in the form of a logic program,
- there is a sound translation of such logic program into an algorithm or artificial neural network, i.e. the obtained algorithm or neural network models given semantics for the logic program,
- there is a learning strategy for obtained algorithm or artificial neural network that allows to obtain "new knowledge",
- there is a sound translation of an algorithm or artificial neural network, that has been changed under the process of learning, back into a logic program,
- it is possible to compare two logic programs and return the symmetric difference between them.

In principle, my procedure for generation of abductive hypotheses will be implemented in artificial neural networks. The initial state would be a knowledge base Γ and an abductive goal ϕ. In the next step an artificial neural network will be created, which represents both: the knowledge base Γ and the abductive goal ϕ. In the next step, the network is trained in such a way, that the abductive goal becomes attainable. The trained neural network represents the modified knowledge base Γ'. The symmetric difference between the initial and the modified knowledge base would be an abductive hypothesis we were seeking.

The following problems emerge when this approach is concerned: how to translate the knowledge base and the abductive goal into the artificial neural network; how to perform learning of the neural network; and how to translate the artificial neural network back to the form of the (modified) knowledge base which could be compared with the initial one. On the one hand, the desired solution would be as follows: the knowledge base, abductive goal and modified knowledge base are introduced in a purely symbolic form. In such case the abductive goal could be defined in terms of the logical consequence. It would be also possible to perform well defined comparison between both knowledge bases. On the other hand, the most common learning strategy for artificial neural networks is the Backpropagation algorithm [5; 14, p. 12; 25, p. 127].

There are a few approaches to the problem of integration of logic and artificial neural networks. Generally speaking, there are two main strategies: the *unified* one and the *hybrid* one ([14, pp. 3–5]; see also [26]). The first one employs neural networks alone, while the second combines neural networks with symbolic models into expert systems, case-based reasoning systems and decision trees [26, pp. 13–14]. The classification of approaches to the neurosymbolic integration proposed by Hilario [26] is depicted in the Figure 1.1.

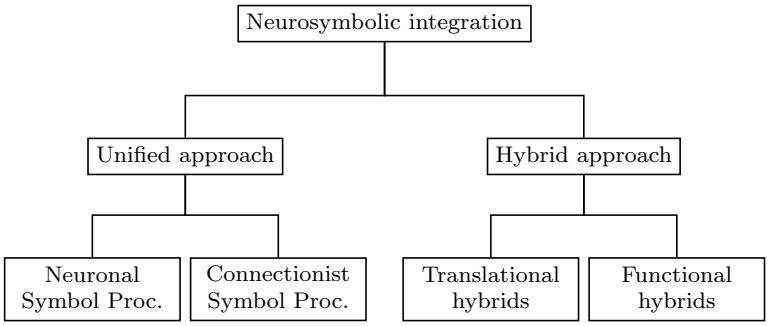

Fig. 1.1 Classification of integrated neurosymbolic systems. [26, p. 14]

Unified strategies can be subdivided into two groups on the basis of the difference in understanding the concept of the neural network. In the neuronal symbol processing group the term *neuronal* denotes the network which properties are as much related to the biological neuronal networks as it is possible. The point of interest is in the simulation of biological neuronal structures and in applying such constructions to cognitive phenomena, e.g. sensorimotor responses, concept formation, language and even higher cognitive functions [26, p. 15]. The approach is bottom-up in the sense that the neural networks are not predefined. The best known representative is the neural Darwinism approach by Gerald Edelman [8], where the network is modified on the basis of the following rules: developmental selection, experiential selection and reentrant mapping [26, p. 15]. An example of this category is Darwin III system [44, 45].

The connectionist symbol processing group makes use of neural networks, where the term *neural* implies that the network is similar to the biological structures only in general. The aim here is to employ the artificial neural networks as building blocks for "cognitive architecture capable of complex symbol processing" [26, pp. 15–16]. In contrast with the former strategy, the connectionist symbol processing approach is top-down rather than bottom-up [26, p. 15–16]. It could be the case that the network does not have the predetermined symbolic structure and processes emerge from it in somehow unforeseen way [26, p. 16]. However, the majority of applications exploit the possibility of establishing such structure of the neural network that is related to the functions the neural network is supposed to model. Systems that are organised in such a way are for example BoltzCONS [53], Dcps [54], Rubicon [52] or CHCL [27].

Hybrid strategies also contain two subgroups: translational and functional hybrids. The main assumption of this approach is that the cooperation of the two disjoint systems: symbolic and neural, is the key for constructions which "can attain the full range of cognitive and computational powers" [26, p. 17]. The translational hybrids are intermediate class between unified models and functional hybrids [26, p. 6]. In the translational hybrids approach only ar-

tificial neural networks are used as processors while the symbolic structures are translated to or obtained from (or both) neural networks, for example system that uses propositional rules [55], fuzzy rules [23] or rules with probability ratings [11]. The functional hybrids approach explores the full range of the cooperation between the artificial neural networks and symbolic rules of inference [26, p. 18]. The variety of possible solutions given by the functional hybrids spreads on two dimensions: the degree of integration, which is a quantitative dimension; and the integration mode determining the way the relation between neural and symbolic components is organised in the whole system, which is a qualitative dimension. We can include in this group distributed neural networks [41], neural networks combined with rule-based approach [22] or WATTS system [33] for wastewater treatment.

In view of the desired properties of the system performing the abductive procedure the most suitable approach is the translational hybrid. Garcez et al. [14] proposed approach that combines features of connectionist logic systems and translational hybrids systems. In this approach a logic program is translated into a neural network, then the obtained neural network is trained by means of Backpropagation algorithm and finally, the trained neural network is translated back into the *new* logic program. The authors claim that this is a new category of *neural-symbolic learning systems*, which should be added to the classification proposed by Hilario (see Figure 1.1). Figure 1.2 depicts the full schema of neural-symbolic learning systems with six main phases [14, p. 6]:

1. Symbolic background knowledge is translated into the initial architecture of a neural network by a *Translation Algorithm*.
2. The assumption is that the background knowledge may not be complete, therefore the neural network is trained on the examples by means of the Backpropagation algorithm.
3. The proved correctness of the *Translation Algorithm* guarantees that the neural network can be used as a massively parallel computational model of the logical consequences in the form of atoms of the theory encoded in it.
4. Step 4 allows to introduce changes in the neural network for better representation of the background knowledge and the training examples (e.g. solving inconsistencies between background knowledge and the examples).
5. Provably correct *Extraction Algorithm* translates all the rules encoded in the trained neural network back to the symbolic representation.
6. This step represents the verification of the obtained knowledge by an expert. It could be the case, that the 'final form' of the background knowledge is treated as a new input for the whole system.

Garcez et al. [14] proposed Translation Algorithms for definite, general and extended logic programs into the three layer partially recurrent neural networks. They also created Extraction Algorithm for three layer neural networks. The most important feature of those algorithms is their provable

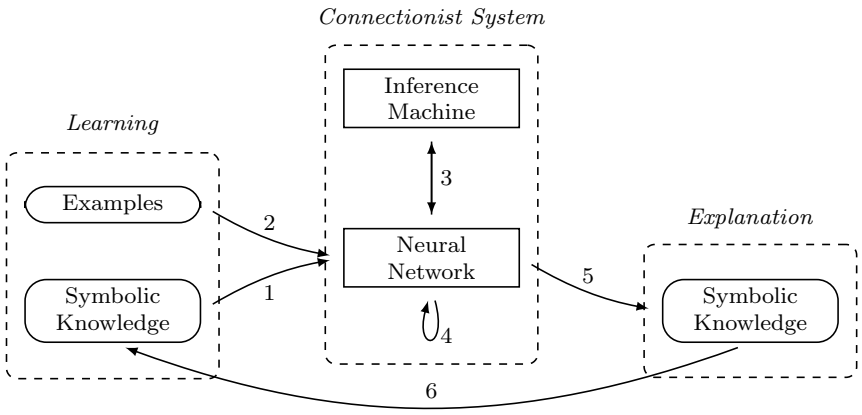

Fig. 1.2 Schema of the neural-symbolic learning systems. [14, p. 6]

correctness. It means, that on the one hand the neural network obtained by means of the Translation Algorithm computes the least Herbrand model, stable model and the answer set for each kind of logic programs respectively (answer set semantics is used as it is simpler and more general than iterated fixpoint semantics for logic programs with negation; see [20, 21]). Additionally, the Translation Algorithm creates neural networks which can be trained by Backpropagation algorithm. On the other hand, the Extraction Algorithm extracts all the clauses that can be obtained from a given neural network.

Neural networks obtained by means of the Translation Algorithms work as a $T_{\mathcal{P}}$ operator for logic programs. The operator $T_{\mathcal{P}}$ is a mapping from interpretations of logic programs into interpretations of logic programs. Van Emden and Kowalski [60] defined the fixpoint characterisation of the least Herbrand model for definite logic programs, which is a fixpoint of $T_{\mathcal{P}}$ when it starts from the empty set. Therefore, with the input vector consisting only of -1 values (which describes the situation of empty interpretation, i.e. when all atoms are *false*), the neural network will finally reach a stable state, where its output represents the fixpoint of $T_{\mathcal{P}}$. The input layer consists of neurons that represent atoms occurring in the bodies of Horn clauses in the logic program, while the heads of those clauses are combined with the neurons from the output layer. The translation from input to output layer runs through the hidden layer which contains neurons representing each Horn clause from the logic program. Due to the recurrent connections the stabilisation of the neural network in most cases is obtained after more than one run of the signal from input to output layer. The *stabilisation* of the neural network means that the state of the neurons does not change in time. As it is described in further paragraphs (see Section 2.3 on page 19 and Subsection 2.4.1 on page 45), when the operator $T_{\mathcal{P}}$ starts from the empty interpretation (which means that all the atoms from the logic program are mapped to *false*), then after

finitely many steps it 'reaches' the fixpoint which is the least Herbrand model of the logic program. Analogically, when all the input neurons in the neural network are set to the -1 value (which symbolises the empty interpretation), the neural network sets itself after finitely many cycles in the stable state and neurons that are active in the output layer are those which are associated with atoms from the least Herbrand model for the logic program. This idea is used in generation of the training set for the neural network described in Section 4.2 on page 97.

Garcez et al. [15] proposed two different approaches to the problem of abductive hypotheses generation by means of the Connectionist Inductive Learning and Logic Programming ($C\text{-}IL^2P$) system. Both concern the situation where a neural network representing a knowledge base is already constructed and "[a]bduction, in this setting, can be seen as the process of finding input values that would result in a particular output value if presented to the network" [15, p. 4]. The first approach involves Connectionist Modal Logic [18, 17]. Generally speaking, the idea is that Horn clauses which are the basis for the initial neural network construction, are 'inverted' and each atom from the body of the original Horn clause is preceded with the possibility symbol. As a result the new set of clauses is obtained which serves as a basis for a new neural network. Activation of neurons in the input layer of the new neural network (they were placed in the output layer in the initial neural network) leads to the activation of specific neurons in the output layer. Those are the neurons associated with atoms we were looking for, i.e. that are a possible explanations for the atoms placed in the heads of clauses in the initial logic program.

The second approach proposed in [15] is also described in [43]. Similarly to the previous account, the main goal of the procedure is to find those neurons from the input layer that are responsible for the activation of the neurons in the output layer of the network. The difference here is that in this approach there can be defined a set of integrity constraints, i.e. a set containing clauses of the form $a_1, \ldots, a_2 \rightarrow \bot$. In other words, the set of integrity constraints consists of sets of atoms which must not be *true* at the same time [15, p. 22]. The procedure runs as follows: all combinations of possible activations of the input neurons are performed. If it is the case that for a combination of activated input neurons the output neuron of our interest is active and in the same time the neuron associated with a contradiction symbol is deactivated, the input pattern is saved. The set of all saved combinations of input atoms is the set of abductive hypotheses.

In [16] a system is described that combines abductive reasoning (understood here in terms of abductive logic programming) with inductive learning to evolve (change) requirements specifications in such a way that they no longer violate desired property, i.e. by means of the abductive reasoning the analysis is conveyed to "discover whether a given systems description satisfies a system property and if not, generate appropriate diagnostic information" [16, pp. 1–2], while inductive learning changes the description of

the system to fulfil all required properties. Although $C\text{-}IL^2P$ was one of the main parts of the system, it was not used to generate abductive hypotheses but only used training sets made on their basis, i.e. the abductive procedure based on the approach described in [49, 50] produced a set of atoms that in turn was used to generate a training example for the artificial neural network created by means of the translation algorithm in $C\text{-}IL^2P$.

What should be observed here is that both mentioned systems can generate abductive hypotheses only in the form of atoms. It could be justified given the fact that the consequence relation for logic programs described as the least Herbrand model also refers only to atoms. The approach described in this work is more general in the sense that abductive hypotheses and goals can be introduced in the form of Horn clauses ([59] is an example where law-like statement of similar form to Horn clauses are generated as abductive hypotheses) as well as atoms. Therefore, the concept of consequence relation has to be modified.

Van Emden and Kowalski [60] described the connections between operational and fixpoint semantics, and a proof theory and model theory for logic programs. They also proved that for definite logic programs the least Herbrand model for a logic program is equal to the set of logical consequences of the logic program. At first, the definition of the $T_{\mathcal{P}}$ operator was given. As Lloyd describes it, "$T_{\mathcal{P}}$ provides the link between the declarative and procedural semantics of $\mathcal{P}$" [35, p. 37], where $\mathcal{P}$ denotes the logic program. Afterwards, they proved that models for logic programs can be characterised in terms of $T_{\mathcal{P}}$, namely interpretation I is a model for logic program $\mathcal{P}$ if and only if it is a fixpoint of $T_{\mathcal{P}}$. Finally, in the same paper Van Emden and Kowalski [60] provided theorem for fixpoint characterisation of the least Herbrand Model. What should be noted here is that the least Herbrand model is a set of atoms, therefore the logical consequences of a logical program that can be obtained by means of the operator $T_{\mathcal{P}}$ are of the form of an atom.

As it was mentioned, the assumption is that the abductive procedure presented in this work generates abductive hypotheses for abductive goals in the form of an atom as well as of a Horn clause. The mechanism used for abductive hypotheses generation is the Backpropagation training algorithm applied to a neural network that represents the knowledge base and the abductive goal. Taking into account that neural networks resulting from the translation of a logic program model the way the immediate consequence operator works, I have to extend the concept of logical consequence for Horn clauses.

The schema of our abductive procedure is depicted in the Figure 1.3. The whole process begins with the knowledge base represented by a logic program $\mathcal{P}$ and an abductive goal $\mathcal{G}_{\mathcal{P}}$, which is not derivable from $\mathcal{P}$. In the next step, the translation algorithm $T_{\mathcal{P}\rightarrow\mathfrak{N}}$ translates $\mathcal{P}$ and $\mathcal{G}_{\mathcal{P}}$ into neural network. $T_{\mathcal{P}\rightarrow\mathfrak{N}}$ is a modified version of translation algorithm proposed by Garcez et al. [14]. The network is trained by means of the Backpropagation algorithm with only one training example, i.e. the example where all input atoms are set on the -1 value, while the output atoms are set to 1 value. The input

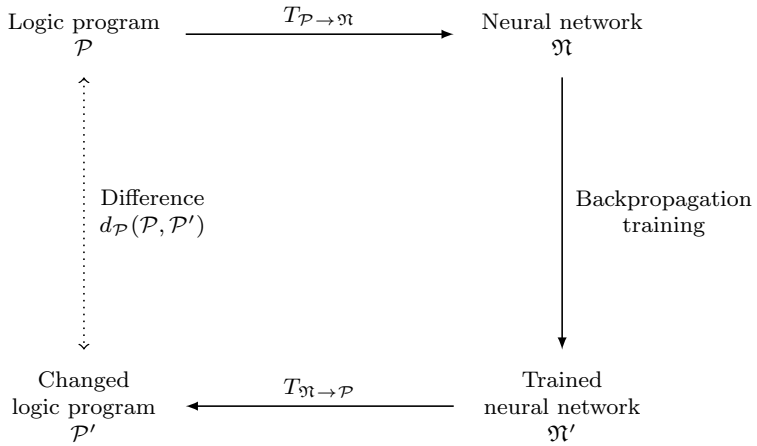

Fig. 1.3 Schema of the abductive procedure.

vector of such form simulates the situation with the empty set as a starting point for $T_{\mathcal{P}}$. The situation is more complicated when the output vector is concerned. On the one hand, setting -1 value for an atom in the output layer prevents it from 'becoming' a fact during the learning process. On the other hand, setting value 1 not necessarily amounts to establishing a fact. It still could be the case, that for a given input, the output neuron would not be active, therefore it would not be a fact, despite the weight of connection with the truth neuron being 'significantly' different than 0 (i.e. a weight which value exceeded the initial limitations for additional connections).

What should be noted here is that the output of the neural network is checked after the stabilisation of the neural network. In this case the situation is analogous to the starting point of the $T_{\mathcal{P}}$ operator with an empty interpretation, and to the final point where the least Herbrand model is identified. After the training (which ends when the error drops below an established level) the trained neural network $\mathfrak{N}$ is translated back to the form of a logic program $\mathcal{P}'$ by means of the extraction algorithm $T_{\mathfrak{N}\to\mathcal{P}}$. $T_{\mathfrak{N}\to\mathcal{P}}$ is also a modified version of extraction algorithm proposed by Garcez et al. [14, ch. 5]. The abductive hypothesis is defined as a symmetric difference $d_{\mathcal{P}}(\mathcal{P},\mathcal{P}')$ between the initial logic program $\mathcal{P}$ and obtained logic program $\mathcal{P}'$.

It is obvious that the difference $d_{\mathcal{P}}$ is not restricted to the addition of information to the initial knowledge base Γ. Therefore there is a possibility that abductive hypothesis would be of the form of modification or removal of information from Γ. As a result, some further considerations are needed in order to distinguish between the formulas which are added to the program and the ones which are removed from it.

Thinking about applications of such abductive procedure on the first place comes the modelling of abductive reasoning on the basis of the results ob-

tained during precisely designed experiments that involve human subjects. The initial conditions given to the subjects can be represented in the form of a logic program and the variety of possibilities for the construction of training sets for the neural networks gives certain flexibility in the abductive hypotheses generation.

The chapters in this work are organised in the following way. The second chapter contains definitions of the consequence relation and abductive goal for definite and normal logic programs. It describes the first property of the abductive procedure that we have listed at the beginning of this chapter, i.e. the symbolic representation of the knowledge base and the abductive goal. The second, third and fourth property of the abductive procedure concerning translation of the formalised theories into a neural networks, training neural network and translating them back into a formalised theory are described in the following two chapters. In the third chapter the concept of the used neural networks is formalised. The integration of symbolic and neural structures is described in the fourth chapter. This is also the chapter where the detailed description of the abductive procedure is given. Additionally, fourth chapter contains a brief description of the preliminary results of the implementation of the abductive procedure in the Framsticks software system [32]. The last chapter before the summary concerns the work that has already been reported in the literature, i.e. Abductive Logic Programming, abductive reasoning modelled in the $C\text{-}IL^2P$ that has been mentioned here in the introductory Chapter and two other approaches that aim at modelling the abductive reasoning, and offers some comparison of the four with approach adopted in this work.

Chapter 2
Logic programs: definite and normal

In this Section the details of definite and normal logic programs will be presented. Different types of logic programs will be introduced in separate sections, where each Section concerning a given type of logic program will end with the definition of an abductive goal for that group of logic programs.

The standard formalisation of logic programs is conveyed in the first order language (see for example [35]). However, the system $C\text{-}IL^2P$ is designed for *grounded* logic programs [14, p. 24–25], i.e. logic programs in which all the variables are instantiated with closed terms. The reason for using only grounded logic programs in $C\text{-}IL^2P$ is that the neurons in input and output layers of the neural network created from a given logic program represent concrete instances of atoms occurring respectively in bodies and heads of clauses in the logic program. As a result, it is justified to employ propositional representation in defining our abductive procedure. This issue is discussed further in the Section 2.1.

Neural networks obtained from logic programs by means of the $C\text{-}IL^2P$ translation algorithms convert the activation of input neurons into activation of output neurons in the same way as the *immediate consequence operator* works for those logic programs [14, p. 43–85]. Therefore, Section 2.2 covers the mathematical background for the construction of the immediate consequence operator.

Sections 2.3 and 2.4 are devoted to the formalisation of definite and normal logic programs, respectively. In both Sections the goal is to describe the method for checking if a given Horn clause is a logical consequence of a logic program, which is based on the notion of immediate consequence operator. Additionally, at the end of the Section 2.4 the problem of finding a fixpoint for immediate consequence operator, as far as normal logic programs are concerned, is described and a family of *acceptable logic programs* is defined.

2.1 Propositional representations

The standard formalisation of logic programs is conveyed in the first order language. A Horn clause in a general form looks as follows:

$$head \leftarrow body \tag{2.1}$$

For definite logic programs the *head* is a single first order atomic formula (*head* cannot be empty) and the *body* consists only of first order atomic formulas separated by commas. In case of normal logic programs the *negation by finite failure* (which will be described in details later on) is allowed before first order atomic formulas in the body of the clause but without iteration. Extended logic programs allow to use classically negated first order atomic formulas in both: the *head* and *body*, along with the negation by finite failure used for first order atomic formulas or classically negated predicates in the *body*. If the *body* is empty, the clause is called a *fact* and the arrow is omitted.

Logic programs are sets of Horn clauses, thus, for example, a definite logic program with two definite Horn clauses could look as follows:

$$\{P(x,y) \leftarrow Q(x), R(y)\ ;\ Q(a)\} \tag{2.2}$$

where x and y are individual variables and a is a constant. The *actual* number of clauses in a logic program depends on the cardinality of the set of the considered objects.

C-IL^2P system is designed for *grounded* logic programs, i.e. logic programs which consist only of *ground instances* of clauses in the logic program [14, p. 24–25]. The process of obtaining ground instances of clauses from a logic program is called *instantiation*: define a fixed language L which includes at least constants occurring in the logic program; the set of ground instances of a clause is the set of clauses obtained by substitution of all the variables occurring in the clause by a term from L (for each variable the same substitution is made for all occurrences of the variable); the grounded logic program is the union of all ground instances [14, p. 24–25].

Grounded logic program obtained from the logic program (2.2) in the previous example looks as follows, when the language L with two constants a and b is concerned:

$$\begin{gathered}\{P(a,a) \leftarrow Q(a), R(a)\ ;\ P(a,b) \leftarrow Q(a), R(b)\ ;\\ P(b,a) \leftarrow Q(b), R(a)\ ;\ P(b,b) \leftarrow Q(b), R(b)\ ;\ Q(a)\}\end{gathered} \tag{2.3}$$

The reason for using only grounded logic programs in *C-IL*^2P system is that the neurons in input and output layer of the neural network created from a given logic program represent concrete instances of atoms occurring respectively in bodies and heads of clauses in the logic program.

After the instantiation all the clauses in the logic program become in fact propositional. The grounded logic program (2.3) can be written as follows:

$$\{a_1 \leftarrow a_2, a_3 \ ; \ a_4 \leftarrow a_2, a_5 \ ; \ a_6 \leftarrow a_7, a_3 \ ; \ a_8 \leftarrow a_7, a_5 \ ; \ a_2 \leftarrow\} \tag{2.4}$$

Only grounded logic programs are taken into consideration in this work. Therefore, the languages used to formalise abductive procedure are propositional ones.

2.2 Mathematical background

Van Emden and Kowalski [60] defined the operational and fixpoint semantics of predicate logic programs. On the one hand, they showed that the operational semantics, which "defines the input-output relation computed by a program in terms of the individual operations evoked by the program inside a machine" [60, p. 733], can be understood as a part of proof theory. On the other hand, the fixpoint semantics can be treated as a special case of model-theoretic semantics. We are especially interested in the fixpoint characterisation of the least Herbrand model, which will be defined in the next Section, and which requires introduction of some mathematical notions. This Section will be organised similarly as in [35, ch. 1, sec. 5] or [14, sec. 2.1, 2.4].

At first we are going to define a *complete lattice*. The reason is that the set of all Herbrand interpretations forms a complete lattice and that fact will be used to define a mapping with properties which are of our concern.

Definition 2.1. 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*. The symbol $\prec$ will be reserved for *strict partial order* relation, i.e. $x \prec y$ if and only if $x \preceq y$ and $x \neq y$.

Definition 2.2. Let S be a set with a partial order $\preceq$ and $x \in S$. We define the following:

- x is a *minimum* of S iff for all elements $y \in S$: $x \preceq y$.
- x is a *maximum* of S iff 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 2.3. 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 a *lower bound* of R if for all elements $y \in R$: $x \preceq y$.
- An element $x \in S$ is the *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 the *greatest lower bound* of R if x is a 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 $lub(R)$ and $glb(R)$ respectively.

Definition 2.4. Let S be a partially ordered set and $R \subseteq S$. We call S a *complete lattice* iff $lub(R)$ and $glb(R)$ exist for every $R \subseteq S$.

Now we are going to describe two properties for mappings defined on complete lattices, which will be needed for our purpose.

Definition 2.5. Let S be a complete lattice and $R \subseteq S$. We call R *directed* iff every finite subset of R has an upper bound in R.

Definition 2.6. Let S be a complete lattice, x and y be elements of S, and $T : S \longrightarrow S$ be a mapping. The following holds:

- T is *monotonic* iff $x \preceq y$ implies $T(x) \preceq T(y)$.
- T is *continuous* iff for every directed subset R of S: $T(lub(R)) = lub(T(R))$.

Fixpoint for a given mapping is such an argument for that mapping that is mapped onto itself. One of the results of Van Emden and Kowalski [60] was that they proved existence of a strict connection between fixpoints of some mappings from and onto set of all Herbrand interpretations of a given logic program and Herbrand models for that logic program.

Definition 2.7. Let S be a complete lattice and $T : S \longrightarrow S$ be a mapping. An element $x \in S$ is the *least fixpoint* of T iff x is a fixpoint of T (i.e. $T(x) = x$) and for every fixpoint y of T: $x \preceq y$. An element $x \in S$ is the *greatest fixpoint* of T iff x is a fixpoint of T and for every fixpoint y of T: $y \preceq x$.

The least fixpoint of T and the greatest fixpoint of T will be denoted as $lfp(T)$ and $gfp(T)$, respectively.

Proposition 2.1. *Let S be a complete lattice and $T : S \longrightarrow S$ be monotonic. Then T has $lfp(T)$ and $gfp(T)$.*

The proof of the Theorem 2.1 is given in [35, p. 28].

Our goal now is to define a way of computing the least fixpoint of a mapping T, assuming that the mapping is monotonic, continuous and defined on a complete lattice. We will use the notion of *ordinal powers of* mapping T. The definition is based on properties of ordinal numbers described in [35, p. 28–29] or [14, p. 26].

Definition 2.8. Let S be a complete lattice and $T : S \longrightarrow S$ be monotonic. Then we define:

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

Proposition 2.2. *Let S be a complete lattice and $T : S \longrightarrow S$ be continuous. Then $\mathit{lfp}(T) = T \uparrow \omega$.*

The ω in Theorem 2.2 denotes the first infinite ordinal. Proof of the Proposition 2.2 is described in details in [35, p. 30].

As we already mentioned (p. 13), the set of all Herbrand interpretations for a logic program forms a complete lattice. Van Emden and Kowalski [60] defined a mapping from and onto the set of all Herbrand interpretations for a logic program, which is called immediate consequence operator. Additionally, immediate consequence operator is monotonic and continuous for definite logic programs and at the end of the next Section we will see that its fixpoints can be associated with Herbrand models for a given logic program. However, the most important property of immediate consequence operator is that by means of definition 2.8 and Proposition 2.2 it will be easy to find its least fixpoint and therefore, the least Herbrand model for a given definite logic program.

Finally, we define a function c, which returns the cardinality of a given set of formulas. It will be used in the following Sections.

Definition 2.9. Let $\mathbb{S}$ be a family of all finite sets of formulas of a considered language and let set $S = \{x, y, \ldots, z\}$ be an element of $\mathbb{S}$. Function $\mathsf{c} : \mathbb{S} \longrightarrow \mathbb{N}$ is defined as follows:

$$\mathsf{c}(S) = \begin{cases} 0 & \text{if } S = \emptyset \\ 1 + \mathsf{c}(\{x, y, \ldots, z\} \setminus \{x\}) & \text{otherwise} \end{cases}$$

2.3 Definite logic programs

In this Section definite logic programs are defined. The formalisation is conveyed similarly to the one given in [14] or [35]. As it was mentioned in the Section 2.1, the language used is a propositional one which keeps the whole formalism simple and minimal. The section ends with the definition of the abductive goal for a definite logic program, where the abductive goal is of the form of an atom or a Herbrand clause. The introduced method of investigation if a given Horn clause is a logical consequence of a definite logic program is based on the immediate consequence operator.

The language $\mathcal{L}_d$ consists of the following elements:

- $\{a_0, a_1, a_2, \ldots\}$ — an infinite, countable set of propositional variables,
- $\leftarrow$ — a primitive connective,
- , — a comma.

Further on we shall omit the reference to $\mathcal{L}_d$, if no ambiguity will arise.

Definition 2.10. The set *Var* of *atomic formulas* consists of all and only propositional variables:

$$Var = \{a_0, a_1, a_2, \ldots\}$$

We shall usually call the elements of the set *Var* just *atoms* or *atomic formulas*.

There is only one kind of non-atomic formulas for definite logic programs and it is called a *Horn clause*.

Definition 2.11. Let a_j, a_k, ..., a_n be atomic formulas. A *Horn clause* h_i is an expression of the form:

$$a_j \leftarrow a_k, \ldots, a_n$$

The atomic formula a_j from the definition of the Horn clause is called the *head* of the Horn clause h_i and will be denoted as $head(h_i)$. The set of atomic formulas $\{a_k, \ldots, a_n\}$ 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. Horn clauses with empty bodies will be called *facts*.

Definition 2.12. The set *Hcl* of Horn clauses consists of all and only Horn clauses:

$$Hcl = \{h_0, h_1, h_2, \ldots\}$$

A *well-formed formula* is either a Horn clause or an atom. We will refer to well-formed formulas as *formulas*.

Definition 2.13. The set *Form* of *well-formed formulas* is defined in the following way:

$$Form = Var \cup Hcl$$

We will now define *definite logic programs*, which are sets of Horn clauses.

Definition 2.14. Let h_i for $1 \leq i \leq n$ be a Horn clause. A *definite logic program* denoted by $\mathcal{P}$ is a finite and non-empty set of Horn clauses:

$$\mathcal{P} = \{h_1, \ldots, h_n\}$$

An exemplary definite logic program with three Horn clauses, where one is a fact may look as follows:

$$\mathcal{P} = \{a_2 \leftarrow a_1, a_4 \ ; \ a_1 \leftarrow a_3 \ ; \ a_5 \leftarrow\} \tag{2.5}$$

To improve readability we will use a semicolon sign ";" instead of a regular comma to separate elements of $\mathcal{P}$.

The following three definitions concern sets of atoms that occur in bodies, heads and bodies or heads of Horn clauses in a given definite logic program, respectively.

Definition 2.15. Let $\mathcal{P}$ be a definite logic program. We define the set of all atoms that occur in bodies of Horn clauses in $\mathcal{P}$, in the following way:

$$B^b_{\mathcal{P}} = \{a_i \mid \text{for some } h_j \in \mathcal{P}\text{: } a_i \in \mathit{body}(h_j)\}$$

Definition 2.16. Let $\mathcal{P}$ be a definite logic program. We define the set of all atoms that occur in heads of Horn clauses in $\mathcal{P}$, in the following way:

$$B^h_{\mathcal{P}} = \{a_i \mid \text{for some } h_j \in \mathcal{P}\text{: } a_i = \mathit{head}(h_j)\}$$

Definition 2.17. Let $\mathcal{P}$ be a definite logic program. We define the set of all atoms that occur in $\mathcal{P}$ in the following way:

$$B_{\mathcal{P}} = B^b_{\mathcal{P}} \cup B^h_{\mathcal{P}}$$

$B_{\mathcal{P}}$ is called a *Herbrand base* of a program $\mathcal{P}$ and is defined as in [35, p. 16], where the language L consists only of constants. Sets $B^b_{\mathcal{P}}$ and $B^h_{\mathcal{P}}$ are defined only for our purposes, although in a similar manner as Herbrand base of a logic program.

We can take under consideration the logic program (2.5) from the previous example. In that case $B^b_{\mathcal{P}}$, $B^h_{\mathcal{P}}$ and $B_{\mathcal{P}}$ look as follows:

$$\begin{aligned} B^b_{\mathcal{P}} &= \{a_1, a_3, a_4\} \\ B^h_{\mathcal{P}} &= \{a_1, a_2, a_5\} \\ B_{\mathcal{P}} &= \{a_1, a_2, a_3, a_4, a_5\} \end{aligned} \tag{2.6}$$

In order to operate on models for definite logic programs we have to define a valuation, which maps formulas into *true* or *false*. Since the language we use in this work is a propositional one, instead of the usually used first order (see Section 2.1), definitions of the valuation and interpretation, and what follows, Corollary 2.1 and 2.2, are contribution of this work.

Definition 2.18. A mapping $v : \mathit{Form} \longrightarrow \{\mathit{true}, \mathit{false}\}$ is a *valuation* defined as follows:

1. For every atomic formula a_i: either $v(a_i) = \mathit{true}$ or $v(a_i) = \mathit{false}$.
2. For every Horn clause h_i: $v(h_i) = \mathit{true}$ iff $v(\mathit{head}(h_i)) = \mathit{true}$ or for at least one $a_j \in \mathit{body}(h_i)$: $v(a_j) = \mathit{false}$.

Instead of checking which atoms are mapped to *true* and which to *false*, we will define an *interpretation* as a set of all those atoms that are mapped to *true*.

Definition 2.19. Let v be a valuation. An *interpretation* w.r.t. v is a set I of all atoms in *Var* that are mapped to *true* by v:

$$I(v) = \{a_i \in \mathit{Var} \mid v(a_i) = \mathit{true}\}$$

Atoms which do not belong to the interpretation $I(v)$ are mapped by valuation v to *false*.

We can easily prove that a Horn clause h_i is mapped by a valuation v to *true* iff the interpretation $I(v)$ contains the head of h_i or there exists an atom a_j in the body of h_i which does not belong to the interpretation $I(v)$.

Corollary 2.1. *Let $I(v)$ be an interpretation w.r.t. a valuation v and h_i a Horn clause. The clause h_i is mapped by v to* true *iff $\mathit{head}(h_i) \in I(v)$ or for at least one $a_j \in \mathit{body}(h_i)$: $a_j \notin I(v)$.*

Proof. ($\rightarrow$) By definition 2.18, h_i is mapped by v to *true*, thus $v(\mathit{head}(h_i)) = \mathit{true}$ or for at least one $a_j \in \mathit{body}(h_i)$: $v(a_j) = \mathit{false}$, and what follows from definition 2.19, $\mathit{head}(h_i) \in I(v)$ or for at least one $a_j \in \mathit{boedy}(h_i)$: $a_j \notin I(v)$.

($\leftarrow$) By definition 2.19, if $\mathit{head}(h_i) \in I(v)$ or for at least one $a_j \in \mathit{body}(h_i)$: $a_j \notin I(v)$, then $v(\mathit{head}(h_i)) = \mathit{true}$ or for at least one $a_j \in \mathit{body}(h_i)$: $v(a_j) = \mathit{false}$, and what follows form definition 2.18, h_i is mapped by v to *true*. □

The second part of Corollary 2.1 can be changed to implication-like form, i.e. *h_i is mapped by v to true iff, if $\mathit{body}(h_i) \subseteq I(v)$, then $\mathit{head}(h_i) \in I(v)$.*

Formulas which are *true* for every interpretation $I(v)$ will be called *tautologies*. For obvious reasons tautologies can be only of the form of Horn clauses.

Definition 2.20. Let h_i be a Horn clause. The clause h_i is called a *tautology* iff it is *true* for every interpretation $I(v)$.

What is more, we can precisely establish types of Horn clauses that are tautologies, i.e. those which contain an atom from their head in their body.

Corollary 2.2. *Let h_i be a Horn clause. The clause h_i is a tautology iff $\mathit{head}(h_i) \in \mathit{body}(h_i)$.*

Proof. ($\rightarrow$) By definition 2.20, if h_i is a tautology, then h_i is *true* for every interpretation $I(v)$ and what follows by Corollary 2.1, for every interpretation $I(v)$: if $\mathit{body}(h_i) \subseteq I(v)$, then $\mathit{head}(h_i) \in I(v)$. In other words, there does not exist an interpretation $I(v)$, where $\mathit{body}(h_i) \subseteq I(v)$ and $\mathit{head}(h_i) \notin I(v)$, therefore $\mathit{head}(h_i) \in \mathit{body}(h_i)$.

($\leftarrow$) For a Horn clause h_i, if $\mathit{head}(h_i) \in \mathit{body}(h_i)$, then for every interpretation $I(v)$, if $\mathit{body}(h_i) \subseteq I(v)$, then $\mathit{head}(h_i) \in I(v)$. Therefore by Corollary 2.1, h_i is *true* for every interpretation $I(v)$ and what follows from definition 2.20, h_i is a tautology. □

Now we are going to define a mapping denoted by $\mathsf{C}_{\mathcal{P}}$, which takes as an argument an interpretation $I(v)$. The mapping $\mathsf{C}_{\mathcal{P}}$ returns heads of all Horn clauses from a definite logic program $\mathcal{P}$, whose bodies belong to the interpretation $I(v)$. It is closely related to the immediate consequence operator defined by Van Emden and Kowalski [60] and described at the end of this Section (see definition 2.34). By $\wp(X)$ we denote the power set of the set X.

Definition 2.21. Let $\mathcal{P}$ be a definite logic program and $I(v)$ an interpretation w.r.t. a valuation v. The mapping $\mathsf{C}_{\mathcal{P}} : \wp(\mathit{Var}) \longrightarrow \wp(\mathit{Var})$ is defined as follows:

$$\mathsf{C}_{\mathcal{P}}(I(v)) =_{df} \{\mathit{head}(h_i) \mid h_i \in \mathcal{P},\ \mathit{body}(h_i) \subseteq I(v)\}$$

Let us consider exemplary definite logic program (2.5) and the following interpretation:

$$\begin{aligned} \mathcal{P} &= \{a_2 \leftarrow a_1, a_4\ ;\ a_1 \leftarrow a_3\ ;\ a_5 \leftarrow\} \\ I(v) &= \{a_1, a_3, a_7\} \end{aligned} \tag{2.7}$$

The mapping $\mathsf{C}_{\mathcal{P}}$ for interpretation $I(v)$ and definite logic program $\mathcal{P}$ from example (2.7) looks as follows:

$$\mathsf{C}_{\mathcal{P}}(I(v)) = \{a_1, a_5\} \tag{2.8}$$

The mapping $\mathsf{C}_{\mathcal{P}}$ enables us to define a mapping called *chain* and denoted by ch. Taking three arguments: a definite logic program $\mathcal{P}$, an interpretation $I(v)$ and an atom a_i, the chain mapping returns 0 or 1, depending on the possibility to "travel" from the interpretation $I(v)$ to the atom a_i through Horn clauses from the definite logic program $\mathcal{P}$. If the information obtained from the mapping ch is 1, we will say that *there is a chain from $I(v)$ to a_i in $\mathcal{P}$*. Although the mapping ch is defined on the basis of mapping $\mathsf{C}_{\mathcal{P}}$, which in turn is based on the immediate consequence operator, the definition of the mapping and its later use in the propositions 2.5, 2.6 and 2.7 is an original contribution of this work.

Definition 2.22. Let $\mathcal{P}$ be a definite logic program, $I(v)$ an interpretation w.r.t. a valuation v and a_i an atom. The mapping ch: $(\mathcal{P}, I(v), a_i) \longrightarrow \{0, 1\}$ called *chain from $I(v)$ to a_i in $\mathcal{P}$* is defined as follows:

$$\mathsf{ch}(\mathcal{P}, I(v), a_i) = \begin{cases} 1 & \text{if } a_i \in I(v) \\ 0 & \text{if } a_i \notin I(v) \text{ and } \mathsf{C}_{\mathcal{P}}(I(v)) \subseteq I(v) \\ \mathsf{ch}(\mathcal{P}, I^*(v), a_i) & \text{otherwise} \end{cases}$$

where $I^*(v) = I(v) \cup \mathsf{C}_{\mathcal{P}}(I(v))$.

For definite logic program $\mathcal{P}$ and interpretation $I(v)$ from example (2.7) the mapping ch will return 1 for the following atoms: a_1, a_3, a_7 and a_5. Let us consider the same definite logic program $\mathcal{P}$ but different interpretation

$I(v_1) = \{a_3, a_4\}$ and an atom a_2. The mapping $\mathsf{ch}(\mathcal{P}, I(v_1), a_2)$ would look as follows:

1. $\mathsf{ch}(\mathcal{P}, \{a_3, a_4\}, a_2) = \mathsf{ch}(\mathcal{P}, \{a_3, a_4, a_1, a_5\}, a_2)$, because:
   a. $a_2 \notin \{a_3, a_4\}$ and
   b. $\mathsf{C}_{\mathcal{P}}(\{a_3, a_4\}) = \{a_1, a_5\}$.
2. $\mathsf{ch}(\mathcal{P}, \{a_3, a_4, a_1, a_5\}, a_2) = \mathsf{ch}(\mathcal{P}, \{a_3, a_4, a_1, a_5, a_2\}, a_2)$, because:
   a. $a_2 \notin \{a_3, a_4, a_1, a_5\}$ and
   b. $\mathsf{C}_{\mathcal{P}}(\{a_3, a_4, a_1, a_5\}) = \{a_1, a_5, a_2\}$.
3. $\mathsf{ch}(\mathcal{P}, \{a_3, a_4, a_1, a_5, a_2\}, a_2) = 1$, because:
   a. $a_2 \in \{a_3, a_4, a_1, a_5, a_2\}$.

What should be noted here is that there will always be a chain from any interpretation $I(v)$ to an atom a_i in definite logic program $\mathcal{P}$, if a_i is the head of fact in $\mathcal{P}$. Additionally, it is obvious that the mapping ch will return 1 for every atom a_i that belongs to the interpretation $I(v)$. We will discuss both these consequences in more details later on.

Interpretations which make all formulas from a given set of formulas *true* will be called *models* for that set. The definition of a model for a set of formulas is the same as in, for example, [35, p. 14].

Definition 2.23. Let S be a set of formulas and $I(v)$ an interpretation w.r.t. a valuation v. A *model for* S (in symbols: $m(S)$) is defined as follows:

$$m(S) =_{df} I(v) \text{ for some valuation } v \text{ such that for every } f \in S\text{: } v(f) = \mathit{true}.$$

We can prove that, when the considered set of formulas S is a singleton with a Horn clause h_i, the interpretation $I(v)$, which is a model $m(S)$ for this set, contains the head of the Horn clause h_i, if it contains the body of Horn clause h_i.

Corollary 2.3. *Let $I(v)$ be an interpretation w.r.t. a valuation v and h_i a Horn clause. $I(v)$ is a model for $\{h_i\}$ iff $head(h_i) \in I(v)$ or for at least one $a_j \in body(h_i) : a_j \notin I(v)$.*

Proof. ($\rightarrow$) By definition 2.23 $I(v)$ is a model for $\{h_i\}$, therefore $v(h_i) = \mathit{true}$ and what follows from Corollary 2.1, $head(h_i) \in I(v)$ or for at least one $a_j \in body(h_i) : a_j \notin I(v)$.

($\leftarrow$) By Corollary 2.1 $head(h_i) \in I(v)$ or for at least one $a_j \in body(h_i) : a_j \notin I(v)$, therefore h_i is mapped to *true* by valuation v and what follows from definition 2.23, interpretation $I(v)$ is a model for set $\{h_i\}$. $\square$

Similarly as in the case of Corollary 2.1, the second part of Corollary 2.3 can be changed into implication-like form, i.e. *$I(v)$ is a model for $\{h_i\}$ iff, if $body(h_i) \subseteq I(v)$, then $head(h_i) \in I(v)$.*

By means of the mapping $\mathsf{C}_{\mathcal{P}}$ we can establish if a given interpretation $I(v)$ is a model for definite logic program $\mathcal{P}$. It could be the case that the mapping $\mathsf{C}_{\mathcal{P}}$ returns the set of atoms equal to the interpretation $I(v)$ for definite logic program $\mathcal{P}$, i.e. $\mathsf{C}_{\mathcal{P}}(I(v)) = I(v)$. According to the definition 2.7 from the previous Section, interpretation $I(v)$ is a fixpoint for mapping $\mathsf{C}_{\mathcal{P}}$. In such situation all heads of Horn clauses from $\mathcal{P}$, whose bodies are in interpretation $I(v)$, are also in interpretation $I(v)$. Therefore, interpretation $I(v)$ is a model for definite logic program $\mathcal{P}$. The following two propositions are a consequence of the fact that mapping $\mathsf{C}_{\mathcal{P}}$ is based on the immediate consequence operator, for which the same properties can be proven [60].

Proposition 2.3. *Let $\mathcal{P}$ be a definite logic program and $I(v)$ an interpretation w.r.t. a valuation v. $I(v)$ is a model for $\mathcal{P}$ iff $\mathsf{C}_{\mathcal{P}}(I(v)) = I(v)$.*

Proof. ($\rightarrow$) By definition 2.23, if interpretation $I(v)$ is a model for $\mathcal{P}$, then for every Horn clause $h_i \in \mathcal{P}$: $v(h_i) = \mathit{true}$ and what follows from Corollary 2.1, for every $h_i \in \mathcal{P}$: if $\mathit{body}(h_i) \subseteq I(v)$, then $\mathit{head}(h_i) \in I(v)$. The last part defines the fixpoint of the mapping $\mathsf{C}_{\mathcal{P}}$, i.e. $\mathsf{C}_{\mathcal{P}}(I(v)) = I(v)$, because it means that for every Horn clause $h_i \in \mathcal{P}$: if $\mathit{body}(h_i) \subseteq I(v)$, then $\mathit{head}(h_i) \in I(v)$.

($\leftarrow$) If $\mathsf{C}_{\mathcal{P}}(I(v)) = I(v)$, then for every Horn clause $h_i \in \mathcal{P}$: if $\mathit{body}(h_i) \subseteq I(v)$, then $\mathit{head}(h_i) \in I(v)$ and what follows from Corollary 2.1, for every $h_i \in \mathcal{P}$: $v(h_i) = \mathit{true}$. Therefore, by definition 2.23 the interpretation $I(v)$ is a model for definite logic program $\mathcal{P}$. $\square$

We can extend Proposition 2.3 on the situation, where mapping $\mathsf{C}_{\mathcal{P}}$ returns a part of the interpretation that was the argument for the mapping $\mathsf{C}_{\mathcal{P}}$, i.e. $\mathsf{C}_{\mathcal{P}}(I(v)) \subseteq I(v)$.

Proposition 2.4. *Let $\mathcal{P}$ be a definite logic program and $I(v)$ an interpretation w.r.t. a valuation v. $I(v)$ is a model for $\mathcal{P}$ iff $\mathsf{C}_{\mathcal{P}}(I(v)) \subseteq I(v)$.*

Proof. ($\rightarrow$) By definition 2.23 interpretation $I(v)$ is a model for $\mathcal{P}$, then for every Horn clause $h_i \in \mathcal{P}$: $v(h_i) = \mathit{true}$ and what follows from Corollary 2.1, for every $h_i \in \mathcal{P}$: if $\mathit{body}(h_i) \subseteq I(v)$, then $\mathit{head}(h_i) \in I(v)$. Therefore, for every Horn clause $h_i \in \mathcal{P}$: if $\mathit{body}(h_i) \subseteq I(v)$, then $\mathit{head}(h_i) \in I(v)$, and what follows, $\mathsf{C}_{\mathcal{P}}(I(v)) \subseteq I(v)$.

($\leftarrow$) If $\mathsf{C}_{\mathcal{P}}(I(v)) \subseteq I(v)$, then for every Horn clause $h_i \in \mathcal{P}$: if $\mathit{body}(h_i) \subseteq I(v)$, then $\mathit{head}(h_i) \in I(v)$ and what follows from Corollary 2.1, for every $h_i \in \mathcal{P}$: $v(h_i) = \mathit{true}$. Therefore, by definition 2.23 the interpretation $I(v)$ is a model for definite logic program $\mathcal{P}$. $\square$

Exemplary models for definite logic program

$$\mathcal{P} = \{a_2 \leftarrow a_1, a_4 \ ; \ a_1 \leftarrow a_3 \ ; \ a_5 \leftarrow\}$$

from (2.5) are the following interpretations:

$$\begin{aligned} I(v_1) &= \{a_5\} \\ I(v_2) &= \{a_1, a_2, a_5\} \\ I(v_3) &= \{a_1, a_5\} \end{aligned} \tag{2.9}$$

It should be noted that if a definite logic program $\mathcal{P}$ contains facts, then every interpretation $I(v)$ which is a model $m(\mathcal{P})$ for $\mathcal{P}$ contains heads of those facts. The reason is obvious, namely the body of a fact is the empty set, i.e. $body(fact) = \emptyset$, which in turn is a subset of any interpretation $I(v)$. This observation leads us to the conclusion that there is no semantic difference between facts and atomic formulas.

Let us focus now on the mapping ch (definition 2.22). We can observe a dependency between the existence of models for a definite logic program $\mathcal{P}$ and a singleton with a Horn clause h_i, and the output value of the mapping $\mathsf{ch}(\mathcal{P}, body(h_i), head(h_i))$. Particularly, if there is no chain from the body of a Horn clause h_i to its head in a definite logic program $\mathcal{P}$, i.e.

$$\mathsf{ch}(\mathcal{P}, body(h_i), head(h_i)) = 0$$

then assuming that the body of a Horn clause h_i is *true* we cannot "travel" to the head of h_i by means of Horn clauses in definite logic program $\mathcal{P}$, and therefore it can be *false*. It is obvious that such Horn clause h_i is not a tautology, because from Corollary 2.2 we have that for a tautology h_i: $head(h_i) \in body(h_i)$ and hence, the mapping $\mathsf{ch}(\mathcal{P}, body(h_i), head(h_i))$ would return 1.

Proposition 2.5. *Let $\mathcal{P}$ be a definite logic program, $m(\mathcal{P})$ a model for $\mathcal{P}$ and h_i a Horn clause. If $\mathsf{ch}(\mathcal{P}, body(h_i), head(h_i)) = 0$, then there exists a model for $m(\mathcal{P})$ $\mathcal{P}$ which is not a model for $\{h_i\}$.*

Proof. By definition 2.22 if $\mathsf{ch}(\mathcal{P}, body(h_i), head(h_i)) = 0$, then $\mathsf{C}_{\mathcal{P}}(I(v)) \subseteq I(v)$ and $head(h_i) \notin I(v)$. By Proposition 2.4 the interpretation $I(v)$ is a model $m(\mathcal{P})$ for definite logic program $\mathcal{P}$. By definition 2.22 $body(h_i) \subseteq I(v)$ and $head(h_i) \notin I(v)$. Therefore by Corollary 2.3, interpretation $I(v)$ is not a model for $\{h_i\}$. $\square$

We can also observe the reverse dependency, i.e. when

$$\mathsf{ch}(\mathcal{P}, body(h_i), head(h_i)) = 1$$

there is no such model $m(\mathcal{P})$ for definite logic program $\mathcal{P}$, which would not be a model for $\{h_i\}$. The intuition is the following: assuming the body of a Horn clause h_i to be *true* we can "travel" by means of Horn clauses in $\mathcal{P}$ to the head of h_i, and hence it will also be *true*. Therefore, every model for definite logic program $\mathcal{P}$ is a model for $\{h_i\}$, because it contains the head of Horn clause h_i, if the body of h_i is in the model.

Proposition 2.6. *Let $\mathcal{P}$ be a definite logic program, $m(\mathcal{P})$ a model of $\mathcal{P}$ and h_i a Horn clause. If $\mathsf{ch}(\mathcal{P}, body(h_i), head(h_i)) = 1$, then for every $m(\mathcal{P})$: $m(\mathcal{P})$ is a model for $\{h_i\}$.*

Proof. By definition 2.22, if $\mathsf{ch}(\mathcal{P}, body(h_i), head(h_i)) = 1$, then the $head(h_i)$ is in $I(v)$. There are two cases we have to consider:

1. $head(h_i) \in body(h_i)$, therefore by Corollary 2.2, Horn clause h_i is a tautology and every model $m(\mathcal{P})$ for $\mathcal{P}$ is a model for $\{h_i\}$.
2. $head(h_i) \notin body(h_i)$, therefore the mapping $\mathsf{C}_{\mathcal{P}}$ returns eventually $head(h_i)$, if $body(h_i) \subseteq I(v)$. In other words, if it is the case that $body(h_i) \subseteq I(v)$ and $\mathsf{C}_{\mathcal{P}}(I(v)) \subseteq I(v)$, then $head(h_i) \in I(v)$. As a consequence of Proposition 2.4 we get that, if $body(h_i) \subseteq m(\mathcal{P})$, then $head(h_i) \in m(\mathcal{P})$. □

Proposition 2.7. *Let $\mathcal{P}$ be a definite logic program and h_i a Horn clause. $\mathsf{ch}(\mathcal{P}, body(h_i), head(h_i)) = 0$ iff there exists $m(\mathcal{P})$ that is not a model for $\{h_i\}$.*

Proof. By propositions 2.5 and 2.6. □

Proposition 2.7 states that by means of the mapping ch we can establish, if there exists a model $m(\mathcal{P})$ for a definite logic program $\mathcal{P}$ which is not a model for a given Horn clause $\{h_i\}$. The Horn clause h_i not necessarily has to be an element of a definite logic program $\mathcal{P}$. Let us consider the following definite logic program $\mathcal{P}$ from example (2.5) and a Horn clause h_i:

$$\begin{aligned} &\mathcal{P} = \{a_2 \leftarrow a_1, a_4 \ ; \ a_1 \leftarrow a_3 \ ; \ a_5 \leftarrow\} \\ &h_i : a_2 \leftarrow a_3, a_4 \end{aligned} \tag{2.10}$$

We have that there is a chain from the body of h_i to the head of h_i in $\mathcal{P}$:

$$\mathsf{ch}(\mathcal{P}, body(h_i), head(h_i)) = 1$$

It is clear that the Horn clause h_i is not an element of definite logic program $\mathcal{P}$. However, there is no such model $m(\mathcal{P})$ for $\mathcal{P}$ which would not be a model for $\{h_i\}$, because whenever atoms from the body of Horn clause h_i are elements of the model $m(\mathcal{P})$ for $\mathcal{P}$, the head of h_i is also an element of that model.

Now we are going to use mapping ch for another purpose. It could be the case that a definite logic program $\mathcal{P}$ consists of two definite logic programs: $\mathcal{P}_1$ and $\mathcal{P}_2$. The model $m(\mathcal{P})$ for $\mathcal{P}$ could be equal to the sum of arbitrary models $m(\mathcal{P}_1)$ and $m(\mathcal{P}_2)$ for $\mathcal{P}_1$ and $\mathcal{P}_2$, respectively, if we add to the $m(\mathcal{P})$ heads of those Horn clauses h_i from $\mathcal{P}$, for which there is a chain from arbitrary interpretation $I(v)$ defined as a subset of $m(\mathcal{P}_1) \cup m(\mathcal{P}_2)$. The following Proposition 2.8 is an original contribution to this work.

Proposition 2.8. *Let $\mathcal{P}_1$ and $\mathcal{P}_2$ be a definite logic programs and $m(\mathcal{P}_1)$ and $m(\mathcal{P}_2)$ models for $\mathcal{P}_1$ and $\mathcal{P}_2$, respectively. If $\mathcal{P} = \mathcal{P}_1 \cup \mathcal{P}_2$, then for every model $m(\mathcal{P})$:*

$$m(\mathcal{P}) = m(\mathcal{P}_1) \cup m(\mathcal{P}_2) \cup \{head(h_i) \mid h_i \in \mathcal{P} \textit{ and for some } I(v) \subseteq m(\mathcal{P}_1) \cup m(\mathcal{P}_2) \textit{:} \ \mathsf{ch}(\mathcal{P}, I(v), head(h_i)) = 1\}$$

Proof. There is only one case we have to consider: when there is an interpretation $I(v) \in m(\mathcal{P}_1) \cup m(\mathcal{P}_2)$, an atom $head(h_i)$ and a chain between them in $\mathcal{P}$: $\mathsf{ch}(\mathcal{P}, I(v), head(h_i)) = 1$. In such situation by Proposition 2.6 $head(h_i) \in m(\mathcal{P})$. Since the proposition assumes that $m(\mathcal{P}_1)$ and $m(\mathcal{P}_2)$ are arbitrary, it could be the case that $head(h_i) \notin m(\mathcal{P}_1) \cup m(\mathcal{P}_2)$ and therefore, we have to add $head(h_i)$ to $m(\mathcal{P}_1) \cup m(\mathcal{P}_2)$. □

The following example will illustrate the intuitions underlying Proposition 2.8. Suppose that we have three definite logic programs $\mathcal{P}$, $\mathcal{P}_1$ and $\mathcal{P}_2$:

$$\begin{aligned} \mathcal{P}_1 &= \{a_2 \leftarrow a_1, a_4 \ ; \ a_1 \leftarrow a_3 \ ; \ a_5 \leftarrow\} \\ \mathcal{P}_2 &= \{a_1 \leftarrow a_5, a_6\} \\ \mathcal{P} &= \mathcal{P}_1 \cup \mathcal{P}_2 \end{aligned} \tag{2.11}$$

Proposition 2.8 allows us to "create" a model for definite logic program $\mathcal{P}$ from arbitrary models $m(\mathcal{P}_1)$ and $m(\mathcal{P}_2)$ for $\mathcal{P}_1$ and $\mathcal{P}_2$, for example:

$$\begin{aligned} m(\mathcal{P}_1) &= \{a_4, a_5\} \\ m(\mathcal{P}_2) &= \{a_6\} \end{aligned} \tag{2.12}$$

However, the sum of two models $m(\mathcal{P}_1) \cup m(\mathcal{P}_2)$ is not a model for definite logic program $\mathcal{P}$, because of the following:

$$\begin{aligned} \mathsf{ch}(\mathcal{P}, \{a_5, a_6\}, a_1) &= 1 \\ \mathsf{ch}(\mathcal{P}, \{a_4, a_5, a_6\}, a_2) &= 1 \end{aligned} \tag{2.13}$$

Therefore, the model for definite logic program $\mathcal{P}$ will be the following:

$$m(\mathcal{P}) = m(\mathcal{P}_1) \cup m(\mathcal{P}_2) \cup \{a_1, a_2\} \tag{2.14}$$

Now we are going to prove that models for definite logic programs can be restricted to atoms from those definite logic programs. In other words, if a set of atoms $m(\mathcal{P})$ is a model for a definite logic program $\mathcal{P}$, then removing from the set $m(\mathcal{P})$ all those atoms that do not occur in $\mathcal{P}$ (i.e. do not belong to the Herbrand base $B_{\mathcal{P}}$ for $\mathcal{P}$) leaves us a model for $\mathcal{P}$. This proposition will be essential when we define Herbrand models for definite logic programs and then employ that notion instead of models for definite logic programs. The following proposition is analogous as in [35, p. 17], although, it is adapted to match the language used in this work.

Proposition 2.9. *Let $\mathcal{P}$ be a definite logic program, $B_{\mathcal{P}}$ a Herbrand base for $\mathcal{P}$ and $m(\mathcal{P})$ a model for $\mathcal{P}$. Then $m(\mathcal{P}) \setminus \{a_k \mid a_k \notin B_{\mathcal{P}}\}$ is also a model for $\mathcal{P}$.*

Proof. Proof goes by induction on the number of elements of $\mathcal{P}$, i.e. $\mathsf{c}(\mathcal{P})$.

1. $\mathsf{c}(\mathcal{P}) = 1$, therefore $\mathcal{P} = \{h_i\}$ and by Corollary 2.3 we have two cases: $\mathit{head}(h_i) \in m(\mathcal{P})$ or for at least one $a_j \in \mathit{body}(h_i)$: $a_j \notin m(\mathcal{P})$.
   a. If it is the case that $\mathit{head}(h_1) \in m(S)$, then
   $$\mathit{head}(h_i) \in m(S) \setminus \{a_k \mid a_k \notin B_{\mathcal{P}}\}$$
   because $\mathit{head}(h_i) \in B_{\mathcal{P}}$.
   b. If it is the case that for at least one $a_j \in \mathit{body}(h_i)$: $a_j \notin m(S)$, then for at least one $a_j \in \mathit{body}(h_l)$: $a_j \notin m(S) \setminus \{a_k \mid a_k \notin B_{\mathcal{P}}\}$.
2. Assuming that the proposition holds for $\mathsf{c}(\mathcal{P}) = i$, it holds also for $\mathsf{c}(\mathcal{P}^*) = i + 1$, where $\mathcal{P}^* = \mathcal{P} \cup \{h_{i+1}\}$. By Proposition 2.8 for every model $m(\mathcal{P}^*)$ for $\mathcal{P}^*$:
   $$\begin{aligned} m(\mathcal{P}^*) = m(\mathcal{P}) \cup m(\{h_{i+1}\}) \cup \{&\mathit{head}(h_l) \mid h_l \in \mathcal{P}^* \text{ and for some} \\ &I(v) \subseteq m(\mathcal{P}) \cup m(\{h_{i+1}\})\text{: } \mathsf{ch}(\mathcal{P}^*, I(v), \mathit{head}(h_l)) = 1\} \end{aligned}$$
   where $m(\mathcal{P})$ is a model for $\mathcal{P}$. $B_{\mathcal{P}^*} = B_{\mathcal{P}} \cup B_{\{h_{i+1}\}}$, therefore we have to show that
   $$m(\{h_{i+1}\}) \setminus \left\{a_k \mid a_k \notin B_{\{h_{i+1}\}}\right\}$$
   is a model for $\{h_{i+1}\}$, because
   $$m(\mathcal{P}) \setminus \{a_k \mid a_k \notin B_{\mathcal{P}}\}$$
   by inductive assumption is a model for $\mathcal{P}$ and the set
   $$\begin{aligned} \{\mathit{head}(h_l) \mid h_l \in \mathcal{P}^* \text{ and for some } I(v) \subseteq m(\mathcal{P}) \cup m(\{h_{i+1}\})\text{:} \\ \mathsf{ch}(\mathcal{P}^*, I(v), \mathit{head}(h_l)) = 1\} \end{aligned}$$
   contains only atoms from $B_{\mathcal{P}^*}$. By Corollary 2.3 we have two cases: $\mathit{head}(h_{i+1}) \in m(\{h_{i+1}\})$ or for at least one $a_j \in \mathit{body}(h_{i+1})$: $a_j \notin m(\{h_{i+1}\})$.
   a. If it is the case that $\mathit{head}(h_{i+1}) \in m(\{h_{i+1}\})$, then
   $$\mathit{head}(h_{i+1}) \in m(\{h_{i+1}\}) \setminus \left\{a_k \mid a_k \notin B_{\{h_{i+1}\}}\right\}$$
   because $\mathit{head}(h_{i+1}) \in B_{\{h_{i+1}\}}$.
   b. If it is the case that for at least one $a_j \in \mathit{body}(h_{i+1})$: $a_j \notin m(\{h_{i+1}\})$, then for at least one $a_j \in \mathit{body}(h_{i+1})$: $a_j \notin m(\{h_{i+1}\}) \setminus \left\{a_k \mid a_k \notin B_{\{h_{i+1}\}}\right\}$.
   □

Looking back at the definite logic program $\mathcal{P}$ from the example (2.5) we can see that in fact any model $m(\mathcal{P})$ for $\mathcal{P}$ reduced to the atoms from Herbrand base $B_{\mathcal{P}}$ (2.6) for $\mathcal{P}$ is also a model for $\mathcal{P}$:

$$\begin{array}{lcl} m(\mathcal{P}) = \{a_5, a_6, a_7, a_8\} & & m(\mathcal{P}) = \{a_5\} \\ m(\mathcal{P}) = \{a_1, a_2, a_5, a_7\} & \xrightarrow[\text{to atoms from } B_{\mathcal{P}}]{\text{reduction}} & m(\mathcal{P}) = \{a_1, a_2, a_5\} \\ m(\mathcal{P}) = \{a_1, a_5, a_{10}\} & & m(\mathcal{P}) = \{a_1, a_5\} \end{array}$$

Now we are going to define a *logical consequence* of a set of formulas.

Definition 2.24. Let S be a set of formulas and f be a formula. We say that f is a *logical consequence of* S iff, for every interpretation $I(v)$, if $I(v)$ is a model for S, then $I(v)$ is a model for $\{f\}$.

The fact that a formula f is a logical consequence of a set of formulas S will be denoted by:

$$S \vDash f$$

Proposition 2.9 states that models for definite logic programs can be restricted only to atoms which occur in those definite logic programs. Such models will be called *Herbrand models*. Herbrand models are specific interpretations, therefore firstly we are going to define interpretation restricted to the atoms that occur in a given definite logic program, called *Herbrand interpretations*. Those definitions are taken form [35, p. 16].

Definition 2.25. Let $\mathcal{P}$ be a definite logic program, $B_{\mathcal{P}}$ a Herbrand base for $\mathcal{P}$ and v a valuation. A *Herbrand interpretation for a program* $\mathcal{P}$ w.r.t. v is a set $I_{\mathcal{P}}$ of all atoms in $B_{\mathcal{P}}$ that are mapped to *true* by v:

$$I_{\mathcal{P}}(v) =_{df} \{a_i \in B_{\mathcal{P}} \mid v(a_i) = \mathit{true}\}$$

All three interpretations from example (2.9) are Herbrand interpretations for definite logic program $\mathcal{P}$ (2.5):

$$\begin{array}{ll} I(v_1) = \{a_5\} & = I_{\mathcal{P}}(v_1) \\ I(v_2) = \{a_1, a_2, a_5\} & = I_{\mathcal{P}}(v_2) \\ I(v_3) = \{a_1, a_5\} & = I_{\mathcal{P}}(v_3) \end{array} \tag{2.15}$$

Herbrand interpretations from the example above (2.15) are also *Herbrand models for* definite logic program $\mathcal{P}$, i.e. all Horn clauses from $\mathcal{P}$ are *true* in those interpretations.

Definition 2.26. 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 for $\mathcal{P}$ w.r.t. valuation v. A *Herbrand model for* $\mathcal{P}$ (in symbols: $m_{\mathcal{P}}$) is defined as follows:

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

On the basis of Proposition 2.9, where we showed that models for definite logic programs can be restricted to atoms occurring in those programs, we can prove that if a definite logic program has a model, then it has a Herbrand model.

Proposition 2.10. *Let $\mathcal{P}$ be a definite logic program. If $\mathcal{P}$ has a model, then it has a Herbrand model.*

Proof. By Proposition 2.9 and the fact that $m(\mathcal{P}) \setminus \{a_j \mid a_j \notin B_{\mathcal{P}}\}$ is a Herbrand model. □

As we have mentioned before, there is no semantic difference between facts and atomic formulas, because both kinds of expressions are *true* in Herbrand interpretation $I_{\mathcal{P}}(v)$ iff they belong to $I_{\mathcal{P}}(v)$ (in case of facts we meant their heads).

Corollary 2.4. *Let $\mathcal{P}$ be a definite logic program and h_i a Horn clause which is a fact. Let us further assume that $h_i \in \mathcal{P}$. Then for every model $m_{\mathcal{P}}$ for $\mathcal{P}$: $head(h_i) \in m_{\mathcal{P}}$.*

Proof. By Corollary 2.3 we have that for every $m_{\mathcal{P}}$: $head(h_i) \in m_{\mathcal{P}}$ or for at least one $a_j \in body(h_i)$: $a_j \notin m_{\mathcal{P}}$. $body(h_i) = \emptyset$, therefore for every $m_{\mathcal{P}}$: $head(h_i) \in m_{\mathcal{P}}$. □

Herbrand models for definite logic programs are sets of atoms occurring in those definite logic programs. Therefore we can find *minimal Herbrand models* and the *least Herbrand model*. Both notions are taken from [35, p. 36–37], along with the Proposition 2.11. However, the proof of the Proposition 2.11 is adapted for the language and definitions used in this work.

Definition 2.27. Let $\mathcal{P}$ be a definite logic program and $m_{\mathcal{P}}$ a Herbrand model for $\mathcal{P}$. By $m_{\mathcal{P}}^{\min}$ we denote a *minimal Herbrand model for $\mathcal{P}$* and define it in the following way:

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

Definition 2.28. Let $\mathcal{P}$ be a definite logic program and $m_{\mathcal{P}}$ a model for $\mathcal{P}$. By $M_{\mathcal{P}}$ we denote the *least Herbrand model for $\mathcal{P}$* defined in the following way:

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

Herbrand models for definite logic programs have the following property, which we are going to prove: the intersection of all Herbrand models for a given logic program $\mathcal{P}$ is also a Herbrand model for $\mathcal{P}$ and it is unique. The consequence of this property is that every definite logic program has a least Herbrand model.

Proposition 2.11. *Let $\mathcal{P}$ be a definite logic program and $\{m_{\mathcal{P}i}\}_1^n$ be a non-empty set of Herbrand models for $\mathcal{P}$. Then the intersection of all Herbrand models for $\mathcal{P}$ $\bigcap_{i \in I} m_{\mathcal{P}i}$ is an Herbrand model for $\mathcal{P}$.*

Proof. Proof goes by induction on $\mathsf{c}(\mathcal{P})$ and by induction on $\mathsf{c}(\{m_{\mathcal{P}i}\})$:

1. $\mathsf{c}(\mathcal{P}) = 1$, therefore $\mathcal{P} = \{h_j\}$. Induction on $\mathsf{c}(\{m_{\mathcal{P}i}\})$:
   a. $\mathsf{c}(\{m_{\mathcal{P}i}\}) = 1$, therefore $\{m_{\mathcal{P}i}\} = \{m_{\mathcal{P}1}\}$ and what follows, $\bigcap_{i \in I} m_{\mathcal{P}i} = m_{\mathcal{P}1}$.
   b. Assuming that the proposition holds for $\mathsf{c}(\{m_{\mathcal{P}i}\}) = k$, we show that it holds for $\mathsf{c}(\{m_{\mathcal{P}i}\}) = k + 1$. By assumption $\bigcap_{i \in I} m_{\mathcal{P}i}$, where $i = k$ is a Herbrand model for $\mathcal{P}$, therefore we have to show that $\bigcap_{k \in K} m_{\mathcal{P}k} \cap m_{\mathcal{P}k+1}$ is also a Herbrand model. By Corollary 2.3 there are the following three cases we have to consider:
      i. $\mathit{head}(h_i) \in \bigcap_{k \in K} m_{\mathcal{P}k}$ and $\mathit{head}(h_i) \in m_{\mathcal{P}k+1}$, therefore
      $$\mathit{head}(h_i) \in \bigcap_{k \in K} m_{\mathcal{P}k} \cap m_{\mathcal{P}k+1}$$
      ii. For at least one $a_l \in \mathit{body}(h_i)$: $a_l \notin \bigcap_{k \in K} m_{\mathcal{P}k}$, therefore
      $$a_l \notin \bigcap_{k \in K} m_{\mathcal{P}k} \cap m_{\mathcal{P}k+1}$$
      iii. For at least one $a_l \in \mathit{body}(h_i)$: $a_l \notin m_{\mathcal{P}k+1}$, therefore
      $$a_l \notin \bigcap_{k \in K} m_{\mathcal{P}k} \cap m_{\mathcal{P}k+1}$$

2. Assuming that the proposition holds for $\mathsf{c}(\mathcal{P}) = l$ we show that it holds for $\mathsf{c}(\mathcal{P}^*) = l + 1$. Induction on $\mathsf{c}(\{m_{\mathcal{P}i}\})$:
   a. $\mathsf{c}(\{m_{\mathcal{P}i}\}) = 1$, therefore $\{m_{\mathcal{P}i}\} = \{m_{\mathcal{P}1}\}$ and what follows, $\bigcap_i m_{\mathcal{P}i} = m_{\mathcal{P}1}$.
   b. Assuming that the proposition holds for $\mathsf{c}(\{m_{\mathcal{P}i}\}) = j$ we show that it holds for $\mathsf{c}(\{m_{\mathcal{P}i}\}) = j + 1$. We have that $\bigcap_i m_{\mathcal{P}i}$, where $i = j$ is a Herbrand model for $\mathcal{P} = \{h_1, h_2, \ldots, h_l\}$ (from assumption in 2. and 2. (b)). $\mathcal{P}^* = \mathcal{P} \cup \{h_{l+1}\}$, therefore by Proposition 2.8 for every $m_{\mathcal{P}^*}$: $m_{\mathcal{P}^*} = m_{\mathcal{P}} \cup m_{\{h_{l+1}\}}$ or $m_{\mathcal{P}^*} = m_{\mathcal{P}} \cup m_{\{h_{l+1}\}} \cup \{\mathit{head}(h_{l+1})\}$, if it is the case that $\mathit{body}(h_{l+1}) \subseteq m_{\mathcal{P}} \cup m_{\{h_{l+1}\}}$, and what follows, $\bigcap_j m_{\mathcal{P}^*j} = \bigcap_j m_{\mathcal{P}j} \cup \bigcap_j m_{\{h_{l+1}\}}$ or $\bigcap_j m_{\mathcal{P}^*j} = \bigcap_j m_{\mathcal{P}j} \cup \bigcap_j m_{\{h_{l+1}\}} \cup \bigcap_j \{\mathit{head}(h_{l+1})\}$, which leads us to a conclusion that $\bigcap_j m_{\mathcal{P}^*j}$ is also a Herbrand model for $\mathcal{P}^*$. Finally, we have to show that $\bigcap_j m_{\mathcal{P}^*j} \cap m_{\mathcal{P}^*j+1}$ is a Herbrand model for $\mathcal{P}^*$. By Corollary 2.3 for every $h_n \in \mathcal{P}^*$ there are the following three cases we have to consider:
      i. $\mathit{head}(h_n) \in \bigcap_j m_{\mathcal{P}^*j}$ and $\mathit{head}(h_n) \in m_{\mathcal{P}^*j+1}$, therefore $\mathit{head}(h_n) \in \bigcap_j m_{\mathcal{P}^*j} \cap m_{\mathcal{P}^*j+1}$.
      ii. For at least one $a_k \in \mathit{body}(h_n)$: $a_k \notin \bigcap_j m_{\mathcal{P}^*j}$, therefore $a_k \notin \bigcap_j m_{\mathcal{P}^*j} \cap m_{\mathcal{P}^*j+1}$.
      iii. For at least one $a_k \in \mathit{body}(h_n)$: $a_k \notin m_{\mathcal{P}^*j+1}$, therefore $a_k \notin \bigcap_j m_{\mathcal{P}^*j} \cap m_{\mathcal{P}^*j+1}$. $\square$

Corollary 2.5. *Let $\mathcal{P}$ be a definite logic program, $\bigcap_{i \in I} m_{\mathcal{P}i}$ an intersection of all Herbrand models for $\mathcal{P}$ and $M_{\mathcal{P}}$ the least Herbrand model for $\mathcal{P}$. Then $M_{\mathcal{P}} = \bigcap_{i \in I} m_{\mathcal{P}i}$.*

Proof. There is only one intersection of all Herbrand models for $\mathcal{P}$ and it fulfils the definition 2.28. □

For definite logic program $\mathcal{P}$ from example (2.5):

$$\mathcal{P} = \{a_2 \leftarrow a_1, a_4 \ ; \ a_1 \leftarrow a_3 \ ; \ a_5 \leftarrow\}$$

the least Herbrand model $M_{\mathcal{P}} = \{a_5\}$.

Our purpose is to modify a definite logic program $\mathcal{P}$ in such a way that will allow us to check, if a given Horn clause h_i is a logical consequence of $\mathcal{P}$, by means of the least Herbrand model of the modified definite logic program $\mathcal{P}$. This solution will be discussed later on. Now we have to introduce two mappings, which allow us to mark and unmark atoms, and therefore to recognise those atoms that are inserted into a modified version of $\mathcal{P}$. Definitions 2.29–2.32 along with Corollary 2.6 and 2.7 are original contribution of this work.

Definition 2.29. Let $\mathbb{S}$ be a family of all finite sets of atoms and set $S = \{a_0, \ldots, a_n\}$ be an element of $\mathbb{S}$. Mapping $\mathsf{m} : (\mathbb{S}, m) \longrightarrow \mathbb{S}$ called *marking* is defined as follows:

$$\mathsf{m}(S, m) = \begin{cases} \emptyset & \text{if } S = \emptyset \\ \{a_0^m\} \cup \mathsf{m}(S \setminus a_0, m) & \text{otherwise} \end{cases}$$

Definition 2.30. Let $\mathbb{S}$ be a family of all finite sets of atoms and set $S = \{a_0^m, \ldots, a_n^m\}$ be an element of $\mathbb{S}$. Mapping $\mathsf{um} : (\mathbb{S}, m) \longrightarrow \mathbb{S}$ called *unmarking* is defined as follows:

$$\mathsf{um}(S, m) = \begin{cases} \emptyset & \text{if } S = \emptyset \\ \{a_0^n\} \cup \mathsf{um}(S \setminus a_0^n, m) & \text{if } n \neq m \\ \{a_0\} \cup \mathsf{um}(S \setminus a_0^m, m) & \text{otherwise} \end{cases}$$

The mapping m adds a given letter to the upper index of every atom from a given set of atoms, while the mapping um removes it. Let us assume that the set of atoms is $S = \{a_1, a_2, a_3\}$ and the letter we want to mark atoms from S with is z. Then both mappings look as follows:

$$\mathsf{m}(S, z) = S^z = \{a_1^z, a_2^z, a_3^z\}$$
$$\mathsf{um}(S^z, z) = S = \{a_1, a_2, a_3\}$$

Marking of a set of atoms will be used to identify which atoms from a model for a given modified definite logic program $\mathcal{P}$ were inserted in the form of assumed facts. At the same time we would like marked atoms to have the

same properties as their unmarked versions, i.e. we would like those atoms to be "invisible" for a valuation v. If, for every Horn clause form modified $\mathcal{P}$, a Herbrand interpretation $I_{\mathcal{P}}(v)$ contains (possibly marked) atoms which after unmarking are equal to the body of a Horn clause from modified $\mathcal{P}$, then $I_{\mathcal{P}}(v)$ is a model for modified $\mathcal{P}$ when the head of every such clause is an element of $I_{\mathcal{P}}(v)$. To achieve this we have to modify the definition of a valuation v in such a way, that it stays the same for unmarked atoms, but additionally takes into account also marked atoms.

Definition 2.31. Let *Form* be the set of all well-formed formulas and $\mathsf{m}(\mathit{Form}, m)$ the set of all marked well-formed formulas, where m is an arbitrary marking string. A mapping $v^* : \mathit{Form} \cup \mathsf{m}(\mathit{Form}, m) \longrightarrow \{\mathit{true}, \mathit{false}\}$ is a *valuation* defined as follows:

1. For every atomic formula a_i: either $v(a_i) = \mathit{true}$ or $v(a_i) = \mathit{false}$.
2. For every Horn clause h_i: $v(h_i) = \mathit{true}$ iff
   - $v(\mathit{head}(h_i)) = \mathit{true}$ or $v(\mathsf{m}(\mathit{head}(h_i), m)) = \mathit{true}$, or
   - for at least one $a_j \in \mathit{body}(h_i)$: $v(a_j) = \mathit{false}$ and $v(\mathsf{m}(a_j, m)) = \mathit{false}$.
3. For every marked atomic formula a_i^m: either $v(a_i^m) = \mathit{true}$ or $v(a_i^m) = \mathit{false}$,

We will now introduce a *modification of a definite logic program $\mathcal{P}$ w.r.t. a Horn clause h_i.*

Definition 2.32. Let $\mathcal{P}$ be a definite logic program and h_i a Horn clause. $\mathcal{P}^{h_i}$ is a modification of $\mathcal{P}$ w.r.t. h_i defined as follows:

$$\mathcal{P}^{h_i} =_{df} \mathcal{P} \cup \{\mathit{fact} \mid \mathit{head}(\mathit{fact}) = a_j;\ a_j \in \mathsf{m}(\mathit{body}(h_i), h)\}$$

In other words, a modified definite logic program $\mathcal{P}^{h_i}$ contains all Horn clauses from definite logic program $\mathcal{P}$ together with all atoms from the body of Horn clause h_i in the form of facts. Let us take for example a definite logic program $\mathcal{P}$ and a Horn clause h_i from (2.10):

$$\mathcal{P} = \{a_2 \leftarrow a_1, a_4\ ;\ a_1 \leftarrow a_3\ ;\ a_5 \leftarrow\}$$
$$h_i := a_2 \leftarrow a_3, a_4$$

Modified definite logic program $\mathcal{P}^{h_i}$ is the following:

$$\mathcal{P}^{h_i} = \left\{a_2 \leftarrow a_1, a_4\ ;\ a_1 \leftarrow a_3\ ;\ a_5 \leftarrow\ ;\ a_3^h \leftarrow\ ;\ a_4^h \leftarrow\right\} \tag{2.16}$$

According to the definition 2.31 the least Herbrand model for definite logic program $\mathcal{P}^{h_i}$ is the following set of atoms:

$$M_{\mathcal{P}^{h_i}} = \left\{a_5, a_3^h, a_4^h, a_1, a_2\right\} \tag{2.17}$$

The modification of a definite logic program $\mathcal{P}$ w.r.t. a Horn clause h_i ($\mathcal{P}^{h_i}$) is a sum of two definite logic programs: the first one is obviously $\mathcal{P}$ and the second one is a definite logic program $\mathcal{P}^*$ which consists only of facts whose heads are marked atoms from the body of h_i. Therefore, a Herbrand model for $\mathcal{P}^{h_i}$ can be defined as a sum of three sets:

1. A Herbrand model for $\mathcal{P}$.
2. The set of marked atoms from the body of h_i (i.e. a Herbrand model for $\mathcal{P}^*$).
3. Heads of those Horn clauses, for which there is a chain from a subset of the sum of two previous sets of atoms (without the marking signs) and the head of Horn clause h_i.

Corollary 2.6. *Let $\mathcal{P}$ be a definite logic program, h_i a Horn clause and $\mathcal{P}^{h_i}$ the modification of $\mathcal{P}$ w.r.t. h_i. Then for every model $m_{\mathcal{P}^{h_i}}$ for $\mathcal{P}^{h_i}$:*

$$m_{\mathcal{P}^{h_i}} = m_{\mathcal{P}} \cup \mathsf{m}(body(h_i), h) \cup \{head(h_j) \mid h_j \in \mathcal{P} \text{ and for some } I(v) \subseteq m_{\mathcal{P}} \cup \mathsf{um}(body(h_i), h)\text{: } \mathsf{ch}(\mathcal{P}, I(v), head(h_j)) = 1\}$$

Proof. By Proposition 2.8 and definition 2.32 for every model $m_{\mathcal{P}^{h_i}}$:

$$m_{\mathcal{P}^{h_i}} = m_{\mathcal{P}} \cup m_{\mathcal{P}^*} \cup \{head(h_j) \mid h_j \in \mathcal{P} \text{ and for some } I(v) \subseteq m_{\mathcal{P}} \cup \mathsf{um}(m_{\mathcal{P}^*}, h)\text{: } \mathsf{ch}(\mathcal{P}^{h_i}, I(v), head(h_j)) = 1\}$$

where

$$\mathcal{P}^* = \{fact \mid head(fact) = a_k;\ a_k \in \mathsf{m}(body(h_i), h)\}$$

By Corollary 2.4:

$$m_{\mathcal{P}^*} = \mathsf{m}(\{head(h_l) \mid h_l \in \mathcal{P}^*\}, h)$$

therefore

$$m_{\mathcal{P}^*} = \mathsf{m}(body(h_i), h)$$

Additionally, since $\mathcal{P}^*$ is a set of facts, the condition for h_j can be restricted to $\mathcal{P}$. □

Corollary 2.6 states that every model for modified definite logic program $\mathcal{P}^{h_i}$ contains the marked body of the Horn clause h_i. Therefore, if it is the case that there is a chain from the body of h_i to its head in $\mathcal{P}$, then the head of h_i is in every model $m_{\mathcal{P}^{h_i}}$ for $\mathcal{P}^{h_i}$ which is reduced by marked body of the h_i. We can prove also the opposite relation, i.e. that, if the head of the Horn clause h_i is an element of every model $m_{\mathcal{P}^{h_i}}$ for modified definite logic program $\mathcal{P}^{h_i}$, then there is a chain form the body of h_i to its head in definite logic program $\mathcal{P}$.

Corollary 2.7. *Let $\mathcal{P}$ be a definite logic program, h_i a Horn clause and $\mathcal{P}^{h_i}$ a modification of $\mathcal{P}$ w.r.t. h_i. $\mathsf{ch}(\mathcal{P}, body(h_i), head(h_i)) = 1$ iff for every model $m_{\mathcal{P}^{h_i}}$ for $\mathcal{P}^{h_i}$: $head(h_i) \in m_{\mathcal{P}^{h_i}} \setminus \mathsf{m}(body(h_i), h)$.*

Proof. ($\rightarrow$) $\mathsf{ch}(\mathcal{P}, body(h_i), head(h_i)) = 1$ and by Corollary 2.6 for every $m_{\mathcal{P}^{h_i}}$:

$$m_{\mathcal{P}^{h_i}} = m_{\mathcal{P}} \cup \mathsf{m}(body(h_i), h) \cup \{head(h_j) \mid h_j \in \mathcal{P} \text{ and for some } I(v) \subseteq m_{\mathcal{P}} \cup \mathsf{um}(body(h_i), h)\text{: } \mathsf{ch}(\mathcal{P}, I(v), head(h_j)) = 1\}$$

Therefore for every $m_{\mathcal{P}^{h_i}}$: $head(h_i) \in m_{\mathcal{P}^{h_i}} \setminus \mathsf{m}(body(h_i), h)$.

($\leftarrow$) For every $m_{\mathcal{P}^{h_i}}$: $head(h_i) \in m_{\mathcal{P}^{h_i}} \setminus \mathsf{m}(body(h_i), h)$, therefore:

1. For at least one $m_{\mathcal{P}^{h_i}}$:

$$head(h_i) \in \{head(h_j) \mid h_j \in \mathcal{P} \text{ and for some } I(v) \subseteq m_{\mathcal{P}} \cup \mathsf{um}(body(h_i), h)\text{: } \mathsf{ch}(\mathcal{P}, I(v), head(h_j)) = 1\}$$

   and what follows $\mathsf{ch}(\mathcal{P}, I(v), head(h_j)) = 1$.
2. For every $m_{\mathcal{P}}$: $head(h_i) \in m_{\mathcal{P}}$ and by Proposition 2.7:

$$\mathsf{ch}(\mathcal{P}, I(v), head(h_j)) = 1$$

□

Finally, we can prove that a Horn clause h_i is a logical consequence of a definite logic program $\mathcal{P}$ iff the head of h_i is an element of the least Herbrand model of the modified definite logic program $\mathcal{P}^{h_i}$.

Theorem 2.1. *Let $\mathcal{P}$ be a definite logic program, h_i a Horn clause, $\mathcal{P}^{h_i}$ a modification of $\mathcal{P}$ w.r.t. h_i and $M_{\mathcal{P}^{h_i}}$ the Herbrand model for $\mathcal{P}^{h_i}$. Then the following holds:*

$$\mathcal{P} \vDash h_i \textit{ iff } head(h_i) \in M_{\mathcal{P}^{h_i}} \setminus \mathsf{m}(body(h_i), h)$$

Proof. ($\rightarrow$) By definition 2.24 $\mathcal{P} \vDash h_i$, therefore for every Herbrand model $m_{\mathcal{P}}$ for $\mathcal{P}$: $m_{\mathcal{P}}$ is a model for $\{h_i\}$ and what follows by Proposition 2.7, there is a chain from the body of h_i to its head in $\mathcal{P}$: $\mathsf{ch}(\mathcal{P}, body(h_i), head(h_i)) = 1$. Thus, by Corollary 2.7 for every model $m_{\mathcal{P}^{h_i}}$ for $\mathcal{P}^{h_i}$: $head(h_i) \in m_{\mathcal{P}^{h_i}} \setminus \mathsf{m}(body(h_i), h)$. Then, by Proposition 2.11

$$head(h_i) \in \bigcap_{j \in J} m_{\mathcal{P}^{h_i} j} \setminus \mathsf{m}(body(h_i), h)$$

and finally by Corollary 2.5 $head(h_i) \in M_{\mathcal{P}^{h_i}} \setminus \mathsf{m}(body(h_i), h)$.

($\leftarrow$) By Corollary 2.5, if $head(h_i) \in M_{\mathcal{P}^{h_i}} \setminus \mathsf{m}(body(h_i), h)$, then

$$head(h_i) \in \bigcap_{j \in J} m_{\mathcal{P}^{h_i} j} \setminus \mathsf{m}(body(h_i), h)$$

and what follows from Proposition 2.11, for every $m_{\mathcal{P}^{h_i}}$: $head(h_i) \in m_{\mathcal{P}^{h_i}} \setminus \mathsf{m}(body(h_i), h)$. Therefore, by Corollary 2.7 $\mathsf{ch}(\mathcal{P}, body(h_i), head(h_i)) = 1$.

Thus, by Proposition 2.7 for every $m_{\mathcal{P}}$: $m_{\mathcal{P}}$ is a model for $\{h_i\}$ and further on by definition 2.24 $\mathcal{P} \vDash h_i$. $\square$

Let us take for example a definite logic program $\mathcal{P}$ and a Horn clause h_i from example (2.10), and the modified $\mathcal{P}$ w.r.t. h_i from example (2.16):

$$\begin{aligned} \mathcal{P} &= \{a_2 \leftarrow a_1, a_4 \ ; \ a_1 \leftarrow a_3 \ ; \ a_5 \leftarrow\} \\ h_i &: a_2 \leftarrow a_3, a_4 \\ \mathcal{P}^{h_i} &= \left\{a_2 \leftarrow a_1, a_4 \ ; \ a_1 \leftarrow a_3 \ ; \ a_5 \leftarrow \ ; \ a_3^h \leftarrow \ ; \ a_4^h \leftarrow\right\} \end{aligned}$$

Following Corollary 2.6 the least Herbrand model $M_{\mathcal{P}^{h_i}}$ for $\mathcal{P}^{h_i}$ can be described as a sum of the three sets of atoms:

$$M_{\mathcal{P}^{h_i}} = \{a_5\} \cup \left\{a_3^h, a_4^h\right\} \cup \{a_1, a_2\} \tag{2.18}$$

The head of the Horn clause h_i is an element of the least Herbrand model for modified definite logic program $\mathcal{P}$, therefore h_i is a logical consequence of $\mathcal{P}$.

On the basis of Theorem 2.1 we are able to define an *abductive goal* $\mathcal{G}_{\mathcal{P}}$ for a definite logic program $\mathcal{P}$ in terms of the least Herbrand model for a modified definite logic program $\mathcal{P}^{h_i}$. In Chapter 1 we said that the abductive goal is understood here as a piece of information which is unattainable from the knowledge base. In this case the konwledge base is a definite logic program $\mathcal{P}$ and the attainability relation is just a logical consequence relation. A piece of information is interpreted as a Horn clause h_i. As we have mentioned before, a valuation mapping v (and also v^*) cannot differentiate between facts and atoms, therefore there would also be no difference between abductive goals in the form of facts and atoms.

Definition 2.33. Let $\mathcal{P}$ be a definite logic program. The Horn clause h_i is called an *abductive goal* for $\mathcal{P}$ (denoted by $\mathcal{G}_{\mathcal{P}}$) iff the following occurs:

$$\mathcal{G}_{\mathcal{P}} =_{df} h_i \text{ such that } \mathcal{P} \nvDash h_i$$

Corollary 2.8. *Let $\mathcal{P}$ be a definite logic program, h_i a Horn clause and $\mathcal{P}^{h_i}$ a modified $\mathcal{P}$ w.r.t. h_i. Let us further assume that $M_{\mathcal{P}^{h_i}}$ is the least Herbrand model for $\mathcal{P}^{h_i}$. Then the Horn clause h_i is called an abductive goal $\mathcal{G}_{\mathcal{P}}$ for $\mathcal{P}$ iff the head of h_i is not an element of the $M_{\mathcal{P}^{h_i}}$.*

Proof. By definition 2.33 and Theorem 2.1. $\square$

At the beginning of this Section we have said that Van Emden and Kowalski [60] proposed a fixpoint semantic for logic programs, which is based on the *immediate consequence operator*. We have also said that neural networks obtained from a given logic program by means of the *C-IL*^{2}*P* translation algorithm emulate the way the immediate consequence operator works for that logic program. Therefore, now we are going to define immediate consequence

operator which takes into account also marked atoms. Afterwards the connection between the least Herbrand model for a given definite logic program and the immediate consequence operator will be described.

The following definition is a modified version of definition from [35, p. 37].

Definition 2.34. Let $\mathcal{P}$ be a definite logic program and $I_{\mathcal{P}}(v^*)$ an interpretation. The mapping $T_{\mathcal{P}}(I_{\mathcal{P}}(v^*)) : \wp(B_{\mathcal{P}}) \longrightarrow \wp(B_{\mathcal{P}})$ called *immediate consequence operator* is defined as follows:

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

The collection of all Herbrand interpretations of a definite logic program $\mathcal{P}$, which is $\wp(B_{\mathcal{P}})$, forms a complete lattice under the partial order of set inclusion. The top and the bottom element of $\wp(B_{\mathcal{P}})$ is $B_{\mathcal{P}}$ and $\emptyset$ respectively. Additionally, we can prove the following properties of the $T_{\mathcal{P}}$ operator (proof of Proposition 2.13 is given in [35, p. 37–38]).

Proposition 2.12. *Let $\mathcal{P}$ be a definite logic program. Then the mapping $T_{\mathcal{P}}$ is monotonic.*

Proof. There is no negation allowed in the bodies of definite logic programs, therefore, if $I_{\mathcal{P}}(v_i^*) \in I_{\mathcal{P}}(v_j^*)$, then $T_{\mathcal{P}}(I(v_i^*)) \in T_{\mathcal{P}}(I(v_j^*))$. $\square$

Proposition 2.13. *Let $\mathcal{P}$ be a definite logic program. Then the mapping $T_{\mathcal{P}}$ is continuous.*

The idea of mapping $\mathsf{C}_{\mathcal{P}}(I(v)) : \wp(Var) \longrightarrow \wp(Var)$ is based on the definition of immediate consequence operator. Therefore, in case of immediate consequence operator we can prove proposition similar to Proposition 2.3 (proof is given in [35, p. 38]).

Proposition 2.14. *Let $\mathcal{P}$ be a definite logic program and $I_{\mathcal{P}}(v^*)$ be an interpretation. Then $I_{\mathcal{P}}(v^*)$ is a model for $\mathcal{P}$ iff $T_{\mathcal{P}}(I_{\mathcal{P}}(v^*)) \subseteq I_{\mathcal{P}}(v^*)$.*

Finally, on the ground of the informations given in the Section 2.2 the following theorem can be proven (proof is given in [35, p. 38]).

Theorem 2.2. *Let $\mathcal{P}$ be a definite logic program. Then $M_{\mathcal{P}} = lfp(T_{\mathcal{P}}) = T_{\mathcal{P}} \uparrow \omega$.*

When we consider modified definite logic program $\mathcal{P}^{h_i}$ from example 2.16 we can calculate the least Herbrand model $M_{\mathcal{P}^{h_i}}$ of $\mathcal{P}^{h_i}$ by means of the immediate consequence operator starting from the emptyset, as it is shown in the following example:

$$T_{\mathcal{P}^{h_i}} \uparrow 0 = \emptyset$$
$$T_{\mathcal{P}^{h_i}} \uparrow 1 = \left\{a_5, a_3^h, a_4^h\right\}$$
$$T_{\mathcal{P}^{h_i}} \uparrow 2 = \left\{a_5, a_3^h, a_4^h, a_1\right\}$$
$$T_{\mathcal{P}^{h_i}} \uparrow 3 = \left\{a_5, a_3^h, a_4^h, a_1, a_2\right\}$$
$$T_{\mathcal{P}^{h_i}} \uparrow 4 = \left\{a_5, a_3^h, a_4^h, a_1, a_2\right\}$$

Interpretation $I(v^*) = \left\{a_5, a_3^h, a_4^h, a_1, a_2\right\}$ is a fixpoint for $T_{\mathcal{P}^{h_i}}$ operator, therefore it is also a model for definite logic program $\mathcal{P}^{h_i}$ and what is more, from Theorem 2.2 we know that it is the least Herbrand model for $\mathcal{P}^{h_i}$.

Looking back at the definition 2.33 of an abductive goal $\mathcal{G}_{\mathcal{P}}$ for a definite logic program $\mathcal{P}$ it is clear that as a consequence of Theorem 2.2 the procedure which checks if a given Horn clause h_i is an abductive goal $\mathcal{G}_{\mathcal{P}}$ for $\mathcal{P}$ can be based on the immediate consequence operator $T_{\mathcal{P}}$. This fact will be extensively used in Chapter 4 where the translation algorithms (from logic programs into neural networks and inversely) are described along with the methods of neural networks training.

2.4 Normal logic programs

In order to allow for more expressive power of logic programs, a negation as a finite failure was allowed before atoms from the bodies of Horn clauses. Sets of such Horn clauses are called *normal logic programs* (sometimes the term of *general logic programs* is used, see e.g. [14]).

The language $\mathcal{L}_g$ we use to define normal logic programs is the language $\mathcal{L}_d$ enriched by the $\sim$ sign denoting *negation as finite failure* (we will refer to $\sim$ just as negation if no ambiguity will arise).

The set of atomic formulas *Var* is defined in the same way as in definition 2.10 in previous Section. Similarly, elements of the set *Var* will be called *atoms* or *atomic formulas* (sometimes we will refer to the set *Var* itself as atoms) and will be denoted by a_i.

We will relate to atoms or their negations as *literals*.

Definition 2.35. Let a_i be an atom. The literal l_i is an atom a_i or its negation $\sim a_i$.

If $l_i = a_j$, then we will say that atom a_j *occurs positively*. If $l_i = \sim a_j$, then we will say that atom a_j *occurs negatively*. Additionally, in both cases we can refer to the *base of a literal* l_i, which is the atom a_j.

Definition 2.36. Let l_i be a literal. The *base of a literal* l_i denoted by $\mathsf{b}(l_i)$ is defined in the following way:

$$\mathsf{b}(l_i) = \begin{cases} a_j & \text{if } l_i = a_j \\ a_j & \text{if } l_i = \sim a_j \end{cases}$$

Definition of normal Horn clauses is a generalised version of definition 2.11 for Horn clauses, therefore we will use the same name in both cases, i.e. just Horn clauses.

Definition 2.37. Let a_j be an atomic formula and $l_k, \ldots, l_n$ be literals. A *Horn clause* h_i is an expression of the form:

$$a_j \leftarrow l_k, \ldots, l_n$$

Similarly as in the previous Section, the atomic formula a_j from the definition 2.37 of a Horn clause is called the *head* of Horn clause h_i and will be denoted as $head(h_i)$. The set of literals $\{l_k, \ldots, l_n\}$ forms the *body* of Horn clause h_i and will be denoted as $body(h_i)$. Horn clauses which do not contain literals in their bodies will be called *facts*.

The body of a Horn clause is defined as a set of literals. It would be also useful to define a set which contains all atoms from the body of a Horn clause, regardless of whether the atoms occur positively or negatively, set which contain all atoms that occur positively and set which contain all atoms that occur negatively in the body of a Horn clause.

Definition 2.38. Let h_i be a Horn clause. The set of atoms that occur in $body(h_i)$ is defined as follows:

$$body^a(h_i) = \{a_j = \mathsf{b}(l_k) \mid l_k \in body(h_i)\}$$

Definition 2.39. Let h_i be a Horn clause. The set of atoms that occur positively in $body(h_i)$ is defined as follows:

$$body^p(h_i) = \{a_j \mid a_j \in body(h_i)\}$$

Definition 2.40. Let h_i be a Horn clause. The set of atoms that occur negatively in $body(h_i)$ is defined as follows:

$$body^n(h_i) = \{a_j \mid \sim a_j \in body(h_i)\}$$

There are two kinds of *well-formed formulas*: Horn clauses and literals. We will refer to well-formed formulas as *formulas*.

Definition 2.41. The set *Form* of *well-formed formulas* is defined in the following way:

$$Form = \{f \mid f = l_j \text{ or } f = h_k\}$$

where l_j is a literal and h_k is a Horn clause.

Definition of a *normal logic program* is the same as definition 2.14 of a definite logic program, i.e. a normal logic program is a set of Horn clauses. Similarly as in the case of normal Horn clauses, normal logic program is a generalisation for definite logic program. Because of this reason we will refer

to normal logic programs as logic programs. An exemplary logic program which is a result of the addition of a Horn clause of the form $a_4 \leftarrow \sim a_6$ to the logic program from example 2.5 looks as follows:

$$\mathcal{P} = \{a_2 \leftarrow a_1, a_4 \ ; \ a_1 \leftarrow a_3 \ ; \ a_5 \leftarrow \ ; \ a_4 \leftarrow \sim a_6\} \tag{2.19}$$

Definition of atoms occurring in bodies of Horn clauses from a logic program $\mathcal{P}$ differs slightly from definition from the previous Section because of the possible presence of negated atoms in the bodies of Horn clauses. The definition of the set of all heads of Horn clauses (denoted by $B^h_{\mathcal{P}}$) from logic program $\mathcal{P}$ is the same as definition 2.16 with the adjustment that it concerns logic programs instead of definite logic programs.

Definition 2.42. Let $\mathcal{P}$ be a logic program. We define the set of all atoms that occur in bodies of Horn clauses in $\mathcal{P}$, in the following way:

$$B^b_{\mathcal{P}} = \{a_i \in \mathit{body}^a(h_j) \mid h_j \in \mathcal{P}\}$$

Herbrand base of a logic program $\mathcal{P}$ containing all atoms that occur in Horn clauses from $\mathcal{P}$ is defined in the same way as in definition 2.17 with the adjustment that it concerns logic programs instead of definite logic programs.

For logic program from example 2.19 sets $B^b_{\mathcal{P}}$, $B^h_{\mathcal{P}}$ and $B_{\mathcal{P}}$ look as follows:

$$\begin{aligned} B^b_{\mathcal{P}} &= \{a_1, a_3, a_4, a_6\} \\ B^h_{\mathcal{P}} &= \{a_1, a_2, a_5\} \\ B_{\mathcal{P}} &= \{a_1, a_2, a_3, a_4, a_5, a_6\} \end{aligned} \tag{2.20}$$

A valuation which maps formulas into *true* or *false* differs from definition 2.18 of valuation from previous Section because now it takes into account also negated atoms.

Definition 2.43. A mapping $v : \mathit{Form} \longrightarrow \{\mathit{true}, \mathit{false}\}$ is a *valuation* defined as follows:

1. For every atomic formula a_i: either $v(a_i) = \mathit{true}$ or $v(a_i) = \mathit{false}$.
2. For every atomic formula a_i: $v(a_i) = \mathit{true}$ iff $v(\sim a_i) = \mathit{false}$.
3. For every Horn clause h_i: $v(h_i) = \mathit{true}$ iff $v(\mathit{head}(h_i)) = \mathit{true}$ or for at least one $l_j \in \mathit{body}(h_i)$: $v(l_j) = \mathit{false}$.

An *interpretation* is a set of all those atoms from the set *Var* that are mapped to *true* by some valuation v. In this case the definition of an interpretation w.r.t. valuation v is the same as definition 2.19. As a straightforward consequence of definition 2.43 we got: an atom a_i belongs to interpretation $I(v)$ iff negated atom a_i, i.e. $\sim a_i$, is mapped to *false* by valuation v.

Corollary 2.9. *Let $I(v)$ be an interpretation w.r.t. a valuation v and a_i be an atom. Atom a_i is an element of interpretation $I(v)$ iff $\sim a_i$ is mapped to* false *by valuation v.*

Proof. ($\rightarrow$) By definition 2.19, atom a_i belongs to interpretation $I(v)$, therefore a_i is mapped to *true* by valuation v and what follows from definition 2.43, the negated atom a_i (i.e. $\sim a_i$) is mapped to *false* by valuation v.

($\leftarrow$) By definition 2.43, $\sim a_i$ is mapped by valuation v to *false*, therefore atom a_i if mapped by v to *true* and following definition 2.19, it belongs to interpretation $I(v)$. $\square$

We can also establish if a Horn clause h_i is *true* or *false* when a given interpretation $I(v)$ is concerned. A Horn clause h_i is mapped by valuation v to *true* iff the interpretation $I(v)$ contains the head of h_i or there exists a literal l_j in the body of h_i which is mapped to *false* by valuation v, therefore for atoms that occur positively in the body of h_i at least one does not belong to $I(v)$, or for atoms that occur negatively in the body of h_i at least one does belong to $I(v)$.

Corollary 2.10. *Let $I(v)$ be an interpretation w.r.t. a valuation v and h_i a Horn clause. The clause h_i is mapped by v to* true *iff $head(h_i) \in I(v)$ or, for at least one $a_j \in body^p(h_i)$: $a_j \notin I(v)$ or for at least one $a_k \in body^n(h_i)$: $a_k \in I(v)$.*

Proof. ($\rightarrow$) By definition 2.43, Horn clause h_i is mapped by valuation v to *true*, therefore $v(head(h_i)) = true$ or for at least one $l_j \in body(h_i)$: $v(l_j) = false$. In the first case, when $v(head(h_i)) = true$, by definition 2.19 $head(h_i) \in I(v)$. In the second case, when for at least one $l_j \in body(h_i)$: $v(l_j) = false$, there are two possibilities:

- $\mathsf{b}(l_j) \in body^p(h_i)$, therefore by definition 2.43 and 2.19 $\mathsf{b}(l_j) \notin I(v)$ or
- $\mathsf{b}(l_j) \in body^n(h_i)$, therefore by definition 2.43 and Corollary 2.9 $\mathsf{b}(l_j) \in I(v)$.

($\leftarrow$) We have to consider two cases. In the first one $head(h_i) \in I(v)$, therefore by definition 2.19 $v(head(h_i)) = true$ and further on by definition 2.43 $v(h_i) = true$. In the second case we have two possibilities:

- for at least one $a_j \in body^p(h_i)$: $a_j \notin I(v)$, therefore by definition 2.19 $v(a_j) = false$ and further on $v(l_m) = false$, where $\mathsf{b}(l_m) = a_j$;
- for at least one $a_k \in body^n(h_i)$: $a_k \in I(v)$, therefore by definition 2.19 $v(a_k) = true$ and further on by Corollary 2.9 $v(l_m) = false$, where $\mathsf{b}(l_m) = a_k$.

As we can see, in both cases the literal l_m from the body of Horn clause h_i is mapped to *false*, therefore by definition 2.43 $v(h_i) = true$. $\square$

The second part of Corollary 2.10 can be changed to implication like form, i.e. *h_i is mapped by v to true iff, if for every $a_j \in body^p(h_i)$: $a_j \in I(v)$ and for every $a_k \in body^n(h_i)$: $a_k \notin I(v)$, then $head(h_i) \in I(v)$.*

Horn clauses that are *true* for every interpretation $I(v)$ will be called *tautologies*, as in the previous Section and definition 2.20 of a tautology can

be applied without any changes. Similarly, we can precisely establish the class of Horn clauses that are tautologies, but in this case to those which contain an atom from their head in their body (occurring positively) or contain in their body an atom and its negated form.

Corollary 2.11. *Let h_i be a Horn clause. The clause h_i is a tautology iff $head(h_i) \in body(h_i)$ or for some atom a_j: $a_j \in body(h_i)$ and $\sim a_j \in body(h_i)$.*

Proof. ($\rightarrow$) If Horn clause h_i is a tautology, then by definition 2.20 h_i is *true* for every interpretation $I(v)$. From Corollary 2.10 we get that for every interpretation $I(v)$: if, for every $a_j \in body^p(h_i)$: $a_j \in I(v)$ and for every $a_k \in body^n(h_i)$: $a_k \notin I(v)$, then $head(h_i) \in I(v)$. Now we have to consider two cases:

- if the first part of implication is fulfilled it is always the case that $head(h_i) \in I(v)$, therefore $head(h_i) \in body^p(h_i)$,
- if the first part of implication is not fulfilled, then the only possibility is that there exists an atom a_l such that $a_l \in body^p(h_i)$ and $a_l \notin I(v)$ or $a_l \in body^n(h_i)$ and $a_l \in I(v)$, therefore $a_l \in body(h_i)$ and $\sim a_l \in body(h_i)$.

($\leftarrow$) There are two cases we have to consider. In the first case $head(h_i) \in body(h_i)$, therefore for every interpretation $I(v)$, if for every $a_j \in body^p(h_i)$: $a_j \in I(v)$ and for every $a_k \in body^n(h_i)$: $a_k \notin I(v)$, then $head(h_i) \in I(v)$. Thus by Corollary 2.10, h_i is *true* for every interpretation $I(v)$ and what follows from definition 2.20, h_i is a tautology.

In the second case for some atom a_j: $a_j \in body(h_i)$ and $\sim a_j \in body(h_i)$. By definition 2.43 and 2.19 for every interpretation $I(v)$: either $a_j \in I(v)$ or $a_j \notin I(v)$. If $a_j \in I(v)$, then by Corollary 2.9 $v(\sim a) = false$ and further on, by definition 2.43 $v(h_i) = true$. If $a_j \notin I(v)$, then by definition 2.19 $v(a_j) = false$ and further on, by definition 2.43 $v(h_i) = true$. Therefore, in either case Horn clause h_i is mapped to *true*, therefore by definition 2.20 h_i is a tautology. $\square$

Definition of a model for a set of formulas m_S is the same as definition 2.23. A logical consequence in the form of a formula f of a set of formulas S is defined in the same way as in definition 2.24 and denoted by:

$$S \models f$$

Definitions of a Herbrand interpretation $I_{\mathcal{P}}(v)$, Herbrand model $m_{\mathcal{P}}$, minimal model $m_{\mathcal{P}}^{\min}$, least Herbrand model $M_{\mathcal{P}}$ for a logic program $\mathcal{P}$ and marking mapping are the same as definitions 2.25, 2.26, 2.27, 2.28, 2.29, respectively. It is straightforward to show that also Proposition 2.10 holds.

In order to check if a given Horn clause h_i is a logical consequence of a logical program $\mathcal{P}$ we are going firstly to modify $\mathcal{P}$ in such a way that the negated atoms in the bodies of Horn clauses from $\mathcal{P}$ are converted to atoms not preceded by negation but with the upper index "n".

Definition 2.44. Let h_i be a Horn clause. Mapping $\mathsf{dn} : h_i \longrightarrow h_j$ is defined as follows:

$$\mathsf{dn}(h_i) = \begin{cases} h_i & \text{if } \mathit{body}^n(h_i) = \emptyset \\ h_j & \text{otherwise} \end{cases}$$

where:

- $\mathit{head}(h_j) = \mathit{head}(h_i)$,
- $\mathit{body}(h_j) = \mathit{body}^p(h_i) \cup \mathsf{m}(\mathit{body}^n(h_i), n)$.

Let us take for example the following Horn clause and its modified version:

$$h_1 : a_1 \leftarrow a_2, a_3, \sim a_4, \sim a_5$$
$$\mathsf{dn}(h_1) = a_1 \leftarrow a_2, a_3, a_4^n, a_5^n$$

Mapping dn converts Horn clauses into definite Horn clauses, where atoms previously preceded by negation are marked with upper index "n". Horn clauses that do not contain negated atoms in their bodies are left unchanged. We can prove that interpretations which are models for Horn clauses are also models for modified Horn clauses with negation converted to the upper index, when we bound ourselves to atoms that are not marked with "n".

Corollary 2.12. *Let h_i be a Horn clause and $I(v)$ an interpretation restricted to atoms that are not marked with "n". If $I(v)$ is a model for h_i, then it is a model for $\mathsf{dn}(h_i)$.*

Proof. By Corollary 2.10 there are three cases we have to consider, when $I(v)$ is a model for h_i:

1. $\mathit{head}(h_i) \in I(v)$.
2. For some $a_j \in \mathit{body}^p(h_i)$: $a_j \notin I(v)$.
3. For some $a_k \in \mathit{body}^n(h_i)$: $a_k \in I(v)$.

From definition 2.44 we have that $\mathit{head}(h_i) = \mathit{head}(\mathsf{dn}(h_i))$ and $\mathit{body}^p(h_i) \subseteq \mathit{body}(\mathsf{dn}(h_i))$, therefore in the first and the second case $I(v)$ is also a model for $\mathsf{dn}(h_i)$.

If it is the third case that occurs, then it means that $\mathit{body}^n(h_i) \neq \emptyset$. Therefore by definition 2.44, Horn clause $\mathsf{dn}(h_i)$ contains atoms marked with "n". We know that interpretation $I(v)$ is restricted to atoms that are not marked with "n", therefore $I(v)$ is a model for $\mathsf{dn}(h_i)$. □

We can also prove that interpretation $I(v)$ which is not a model for a Horn clause h_i is also not a model for modified Horn clause $\mathsf{dn}(h_i)$, if we enlarge $I(v)$ with the body of $\mathsf{dn}(h_i)$.

Corollary 2.13. *Let h_i be a Horn clause and $I(v)$ an interpretation restricted to atoms that are not marked with "n". If $I(v)$ is not a model for h_i, then $I(v) \cup \mathit{body}(\mathsf{dn}(h_i))$ is not a model for $\mathsf{dn}(h_i)$.*

Proof. From Corollary 2.10 we have the following:

- $head(h_i) \notin I(v)$ and
- $body^p(h_i) \subseteq I(v)$ and
- $body^n(h_i) \cap I(v) = \emptyset$.

From facts above and definition 2.44 we have that

$$I(v) \cup body(\mathsf{dn}(h_i)) = I(v) \cup \mathsf{m}(body^n(h_i), n)$$

We will refer to $I(v) \cup body(\mathsf{dn}(h_i))$ as $I^*(v)$.

Additionally, since $I(v)$ is not a model for Horn clause h_i, then by definition 2.20, h_i is not a tautology.

Now we can establish the following facts:

- $head(\mathsf{dn}(h_i)) \notin I^*(v)$, because from definition 2.44 we have that $head(\mathsf{dn}(h_i)) = head(h_i)$ and we know that:
  - $head(h_i) \notin I(v)$ and
  - $head(h_i) \notin body(h_i)$, since h_i is not a tautology,
- $body(\mathsf{dn}(h_i)) \subseteq I^*(v)$, because from definition 2.44:

  $$body(\mathsf{dn}(h_i)) = body^p(h_i) \cup \mathsf{m}(body^n(h_i), n)$$

  and we have already established that $body^p(h_i) \cup \mathsf{m}(body^n(h_i), n) \subseteq I^*(v)$.

Therefore by Corollary 2.10, $I(v) \cup body(\mathsf{dn}(h_i))$ is not a model for $\mathsf{dn}(h_i)$. □

Now we will define a *positive version of* a logic program $\mathcal{P}$, denoted by $\mathcal{P}_p$, which consists of Horn clauses with atoms marked with the upper index "n" if they are preceded with negation.

Definition 2.45. Let $\mathcal{P}$ be a logic program. A *positive version of* $\mathcal{P}$ is defined as follows:

$$\mathcal{P}_p = \{h_i \mid h_i = \mathsf{dn}(h_j), h_j \in \mathcal{P}\}$$

The positive version of a logic program from example 2.19 looks as follows:

$$\mathcal{P} = \{a_2 \leftarrow a_1, a_4 \ ; \ a_1 \leftarrow a_3 \ ; \ a_5 \leftarrow \ ; \ a_4 \leftarrow \sim a_6\}$$
$$\mathcal{P}_p = \{a_2 \leftarrow a_1, a_4 \ ; \ a_1 \leftarrow a_3 \ ; \ a_5 \leftarrow \ ; \ a_4 \leftarrow a_6^n\}$$

An interpretation $I(v)$, which is a model for logic program $\mathcal{P}$ and is restricted to atoms that are not marked with "n" is also a model for positive version of $\mathcal{P}$.

Corollary 2.14. *Let $\mathcal{P}$ be a logic program, $\mathcal{P}_p$ a positive version of $\mathcal{P}$ and $I(v)$ an interpretation restricted to atoms that are not marked with "n". If $I(v)$ is a model for $\mathcal{P}$, then it is a model for $\mathcal{P}_p$.*

Proof. Interpretation $I(v)$ is restricted to atoms that are not marked with "n" and it is a model for every Horn clause $h_i \in \mathcal{P}$. By definition 2.45 $\mathcal{P}_p$ consists only of Horn clauses of the form $\mathsf{dn}(h_i)$ for every $h_i \in \mathcal{P}$. Therefore by Corollary 2.12, the interpretation $I(v)$ is a model for every Horn clause $h_j \in \mathcal{P}_p$, and thus it is a model for $\mathcal{P}_p$. $\square$

We can also prove that if an interpretation $I(v)$ (restricted to atoms that are not marked with "n") is a model for logic program $\mathcal{P}$ and is not a model for Horn clause h_i (hence $\mathcal{P} \nvDash h_i$), then $I(v)$ extended by atoms from the body of modified Horn clause $\mathsf{dn}(h_i)$ is a model for positive version of $\mathcal{P}$.

Proposition 2.15. *Let $\mathcal{P}$ be a logic program, $\mathcal{P}_p$ a positive version of $\mathcal{P}$, h_i a Horn clause and $I(v)$ an interpretation restricted to atoms that are not marked with "n". If $I(v)$ is a model for $\mathcal{P}$ and is not a model for h_i, then $I(v) \cup \mathit{body}(\mathsf{dn}(h_i))$ is a model for $\mathcal{P}_p$.*

Proof. Taking under consideration definition 2.20 and the fact that interpretation $I(v)$ is not a model for Horn clause h_i, it is clear that h_i cannot be a tautology. Additionally, by Corollary 2.10 we have the following:

- $\mathit{head}(h_i) \notin I(v)$ and
- $\mathit{body}^p(h_i) \subseteq I(v)$ and
- $\mathit{body}^n(h_i) \cap I(v) = \emptyset$.

Therefore by definition 2.44 we have that

$$I(v) \cup \mathit{body}(\mathsf{dn}(h_i)) = I(v) \cup \mathsf{m}(\mathit{body}^n(h_i), n)$$

Now we have to consider two cases:

1. $\mathit{body}^n(h_i) = \emptyset$, therefore $I(v) \cup \mathit{body}(\mathsf{dn}(h_i)) = I(v) \cup \mathsf{m}(\mathit{body}^n(h_i), n) = I(v) \cup \mathsf{m}(\emptyset, n) = I(v)$. Thus, by Corollary 2.14 we have that $I(v) \cup \mathit{body}(\mathsf{dn}(h_i))$ is a model for $\mathcal{P}_p$.
2. $\mathit{body}^n(h_i) \neq \emptyset$ and since interpretation $I(v)$ is a model for $\mathcal{P}$, then by Corollary 2.10 we have to consider the following situations for each Horn clause $h_j \in \mathcal{P}$:

   a. $\mathit{head}(h_j) \in I(v)$.
   b. For some $a_k \in \mathit{body}^p(h_j)$: $a_k \notin I(v)$.
   c. $\mathit{body}^p(h_j) \subseteq I(v)$ and for some $a_l \in \mathit{body}^n(h_j)$: $a_l \in I(v)$.

If it is the case 2a, then by definition 2.44 $\mathit{head}(h_j) = \mathit{head}(\mathsf{dn}(h_j))$, and therefore $I(v) \cup \mathit{body}(\mathsf{dn}(h_i))$ is a model for $\mathsf{dn}(h_j)$, because $\mathit{head}(\mathsf{dn}(h_j)) \in I(v) \cup \mathit{body}(\mathsf{dn}(h_i))$.

If it is the case 2b, then by definition 2.44 atom $a_k \in \mathit{body}(\mathsf{dn}(h_j))$. We know also that $a_k \notin I(v) \cup \mathit{body}(\mathsf{dn}(h_i))$, because we have already established that interpretation $I(v)$ is enriched only by atoms from $\mathit{body}^n(h_i)$ which are marked with "n". Thus $I(v) \cup \mathit{body}(\mathsf{dn}(h_i))$ is also a model for $\mathsf{dn}(h_j)$.

If it is the case 2c, then atom $\mathsf{m}(a_l, n) \notin I(v) \cup \mathit{body}(\mathsf{dn}(h_i))$, because we have established that $I(v) \cup \mathit{body}(\mathsf{dn}(h_i)) = I(v) \cup \mathsf{m}(\mathit{body}^n(h_i))$ and $a_l \notin \mathit{body}^n(h_i)$ for the following reasons: $a_l \in I(v)$ and $\mathit{body}^n(h_i) \cap I(v) = \emptyset$. Therefore, $I(v) \cup \mathit{body}(\mathsf{dn}(h_i))$ is a model for $\mathsf{dn}(h_j)$, because if $a_l \in \mathit{body}^n(h_j)$, then by definition 2.44 $\mathsf{m}(a_l, n) \in \mathit{body}(\mathsf{dn}(h_j))$, and we have established that $\mathsf{m}(a_l, n) \notin I(v) \cup \mathit{body}(\mathsf{dn}(h_i))$.

In this way we have shown that if interpretation $I(v)$ is a model for Horn clause $h_j \in \mathcal{P}$, then interpretation $I(v) \cup \mathit{body}(\mathsf{dn}(h_i))$ is a model for $\mathsf{dn}(h_j)$ and therefore, $I(v) \cup \mathit{body}(\mathsf{dn}(h_i))$ is a model for $\mathcal{P}_p$. □

There is a strict dependency between the existence of a chain from the body of a modified Horn clause h_i and its head in a positive version of a logic program $\mathcal{P}$ and the fact of h_i being a logical consequence of $\mathcal{P}$. Namely, if it is the case that $\mathsf{ch}(\mathcal{P}_p, \mathit{body}(\mathsf{dn}(h_i)), \mathit{head}(h_i)) = 1$, then every model for $\mathcal{P}$ is also a model for h_i.

Proposition 2.16. *Let $\mathcal{P}$ be a logic program, $\mathcal{P}_p$ a positive version of $\mathcal{P}$, h_i a Horn clause and $I(v)$ an interpretation. If $\mathsf{ch}(\mathcal{P}_p, \mathit{body}(\mathsf{dn}(h_i)), \mathit{head}(h_i)) = 1$, then for every $I(v)$: if $I(v)$ is a model for $\mathcal{P}$, then $I(v)$ is a model for h_i.*

Proof. Let us assume the following:

1. $\mathsf{ch}(\mathcal{P}_p, \mathit{body}(\mathsf{dn}(h_i)), \mathit{head}(h_i)) = 1$.
2. There is an interpretation $I(v)$ such that it it is a model for $\mathcal{P}$ and not a model for h_i.

From the first assumption and by Proposition 2.3 we have that there is no $I(v)$ such that $I(v)$ is a model for $\mathcal{P}_p$ and not a model for $\mathsf{dn}(h_i)$.

From the second assumption we have the following:

- $I(v) \cup \mathit{body}(\mathsf{dn}(h_i))$ is a model for $\mathcal{P}_p$, because of Proposition 2.15,
- $I(v) \cup \mathit{body}(\mathsf{dn}(h_i))$ is not a model for $\mathsf{dn}(h_i)$, because of Corollary 2.13.

Therefore, there is an interpretation $I^*(v) = I(v) \cup \mathit{body}(\mathsf{dn}(h_i))$ such that is a model for $\mathcal{P}_p$ and not a model for $\mathsf{dn}(h_i)$, which is contradictory to our first assumption. □

Finally, we can prove the following: if head of a Horn clause h_i belongs to the least Herbrand model of positive version of logic program $\mathcal{P}$ modified w.r.t. $\mathsf{dn}(h_i)$, then h_i is a logical consequence of $\mathcal{P}$.

Theorem 2.3. *Let $\mathcal{P}$ be a logic program and h_i a Horn clause. Then the following occurs:*

$$\textit{if } \mathit{head}(h_i) \in M_{\mathcal{P}_p^{\mathsf{dn}(h_i)}} \textit{ then } \mathcal{P} \vDash h_i$$

Proof. By Theorem 2.1 we have that if $\mathit{head}(h_i) \in M_{\mathcal{P}_p^{\mathsf{dn}(h_i)}}$, then

$$\mathsf{ch}(\mathcal{P}_p, \mathit{body}(\mathsf{dn}(h_i)), \mathit{head}(h_i)) = 1$$

and what follows from Proposition 2.16, $\mathcal{P} \vDash h_i$. □

Let us take for example the following logic program $\mathcal{P}$ and Horn clause h_1:

$$\begin{aligned} &\mathcal{P} = \{a_2 \leftarrow a_1, \sim a_4 \ ; \ a_3 \leftarrow a_1\} \\ &h_1 : \ a_2 \leftarrow a_1, \sim a_4 \end{aligned} \tag{2.21}$$

It is clear that h_1 should be a logical consequence of $\mathcal{P}$, since it is an element of it. To check if it is the case we make positive version of $\mathcal{P}$ and modified h_1:

$$\begin{aligned} &\mathcal{P}_p = \{a_2 \leftarrow a_1, a_4^n \ ; \ a_3 \leftarrow a_1\} \\ &\mathsf{dn}(h_1) : a_2 \leftarrow a_1, a_4^n \end{aligned} \tag{2.22}$$

Now we are able to modify $\mathcal{P}_p$ w.r.t. $\mathsf{dn}(h_1)$:

$$\mathcal{P}_p^{\mathsf{dn}(h_1)} = \left\{a_2 \leftarrow a_1, a_4^n \ ; \ a_3 \leftarrow a_1 \ ; \ a_1^h \leftarrow \ ; \ a_4^{nh} \leftarrow\right\} \tag{2.23}$$

The least Herbrand model for $\mathcal{P}_p^{\mathsf{dn}(h_1)}$ can be found by means of the immediate consequence operator:

$$\begin{aligned} &T_{\mathcal{P}} \uparrow 0 = \emptyset \\ &T_{\mathcal{P}} \uparrow 1 = \left\{a_1^h, a_4^{nh}\right\} \\ &T_{\mathcal{P}} \uparrow 2 = \left\{a_1^h, a_4^{nh}, a_2, a_3\right\} \\ &T_{\mathcal{P}} \uparrow 3 = \left\{a_1^h, a_4^{nh}, a_2, a_3\right\} = M_{\mathcal{P}_p^{\mathsf{dn}(h_1)}} \end{aligned} \tag{2.24}$$

It is the case that $head(h_1) \in M_{\mathcal{P}_p^{\mathsf{dn}(h_1)}}$, therefore Horn clause h_1 is a logical consequence of logic program $\mathcal{P}$.

Described method is not complete, i.e. there can be found such Horn clauses whose head does not belong to the least Herbrand model of the modified positive version of logic program, but they are logical consequences of that program. The reason for the incompleteness is the way the immediate consequence operator works for normal logic programs. The problem is discussed in details in the next Subsection 2.4.1, where the notion of *acceptable logic program* is defined. For now, let us consider the same logic program as in example 2.21 but with a slightly modified Horn clause:

$$\begin{aligned} &\mathcal{P} = \{a_2 \leftarrow a_1, \sim a_4 \ ; \ a_3 \leftarrow a_1\} \\ &h_2 : \ a_2 \leftarrow a_1, \sim a_3 \end{aligned} \tag{2.25}$$

The positive version of $\mathcal{P}$ is the same as in example 2.22, but now modified Horn clause h_2 and what follows, modified positive version of $\mathcal{P}$ w.r.t. $\mathsf{dn}(h_2)$ is different:

$$\begin{aligned} \mathcal{P}_p &= \{a_2 \leftarrow a_1, a_4^n \ ; \ a_3 \leftarrow a_1\} \\ \mathsf{dn}(h_2) &: a_2 \leftarrow a_1, a_3^n \\ \mathcal{P}_p^{\mathsf{dn}(h_2)} &= \left\{a_2 \leftarrow a_1, a_4^n \ ; \ a_3 \leftarrow a_1 \ ; \ a_1^h \leftarrow \ ; \ a_3^{nh} \leftarrow\right\} \end{aligned} \tag{2.26}$$

The least Herbrand model for $\mathcal{P}_p^{\mathsf{dn}(h_2)}$ looks as follows:

$$\begin{aligned} T_{\mathcal{P}} \uparrow 0 &= \emptyset \\ T_{\mathcal{P}} \uparrow 1 &= \left\{a_1^h, a_3^{nh}\right\} \\ T_{\mathcal{P}} \uparrow 2 &= \left\{a_1^h, a_3^{nh}, a_3\right\} \\ T_{\mathcal{P}} \uparrow 3 &= \left\{a_1^h, a_3^{nh}, a_3\right\} = M_{\mathcal{P}_p^{\mathsf{dn}(h_2)}} \end{aligned} \tag{2.27}$$

The head of Horn clause h_2 does not belong to $M_{\mathcal{P}_p^{\mathsf{dn}(h_2)}}$, but there is no such interpretation $I(v)$ which is a model for $\mathcal{P}$ and not a model for h_2, hence $\mathcal{P} \models h_2$.

Definition of abductive goal $\mathcal{G}_{\mathcal{P}}$ for a logic program $\mathcal{P}$ is the same as the definition 2.33 with the adjustment that it concerns normal logic programs instead of definite logic programs. Additionally, by means of the Theorem 2.3 we can prove that if $\mathcal{P} \not\models h_i$, then $head(h_i) \notin M_{\mathcal{P}_p^{\mathsf{dn}(h_i)}}$.

Corollary 2.15. *Let $\mathcal{P}$ be a logic program, h_i a Horn clause and $M_{\mathcal{P}_p^{\mathsf{dn}(h_i)}}$ the least Herbrand model of positive version of $\mathcal{P}$ modified w.r.t. h_i. If $\mathcal{P} \not\models h_i$, then $head(h_i) \notin M_{\mathcal{P}_p^{\mathsf{dn}(h_i)}}$.*

Proof. By Theorem 2.3.

2.4.1 Acceptable normal logic programs

Immediate consequence operator $T_{\mathcal{P}}$ for normal logic programs is not a monotonic mapping. In fact, it is not even continuous. Therefore, there are logic programs for which $T_{\mathcal{P}}$ operator does not have a fixpoint, what results in its "inability to stop". Let us first modify the definition of $T_{\mathcal{P}}$ operator to suit normal logic programs.

Definition 2.46. Let $\mathcal{P}$ be a normal logic program and $I_{\mathcal{P}}(v)$ an interpretation. The mapping $T_{\mathcal{P}} : \wp(B_{\mathcal{P}}) \longrightarrow \wp(B_{\mathcal{P}})$ is defined as follows:

$$\begin{aligned} T_{\mathcal{P}}(I_{\mathcal{P}}(v)) =_{df} \{head(h_i) \mid h_i \in \mathcal{P} \text{ and } body^p(h_i) \subseteq I_{\mathcal{P}}(v) \text{ and} \\ body^n(h_i) \cap I_{\mathcal{P}}(v) = \emptyset\} \end{aligned}$$

Definition 2.46 of immediate consequence operator $T_{\mathcal{P}}$ for normal logic programs is extended version of definition 2.34. Definite Horn clause h_i contains only those atoms that occur positively, therefore $body^n(h_i) = \emptyset$.

Now let us consider the following example of a logic program $\mathcal{P}$:

$$\mathcal{P} = \{a_1 \leftarrow \sim a_1\} \tag{2.28}$$

The immediate consequence operator $T_{\mathcal{P}}$ has no fixpoint, starting from the bottom of $\wp(B_{\mathcal{P}})$:

$$\begin{aligned} &T_{\mathcal{P}} \uparrow 0 = \emptyset \\ &T_{\mathcal{P}} \uparrow 1 = \{a_1\} \\ &T_{\mathcal{P}} \uparrow 2 = \emptyset \\ &\qquad\vdots \end{aligned} \tag{2.29}$$

It is also the case for a neural network obtained from such a logic program that it cannot stabilise itself, because, as we have said (p. 11), such neural networks reflect the way the immediate consequence operator works. Therefore, we will introduce a concept of *acceptable logic programs* for which the immediate consequence operator always reaches a fixpoint.

The following notions are based on definitions introduced by Apt and Pedreschi [3, p. 128–129]. At first we want to establish if a given atom $a_i \in B_{\mathcal{P}}$ *depends positively* or *negatively* on other atoms from Herbrand base $B_{\mathcal{P}}$. In that purpose we build a *graph* $G_{\mathcal{P}}$ for a given logic program $\mathcal{P}$.

Definition 2.47. Let $\mathcal{P}$ be a logic program and $B_{\mathcal{P}}$ the Herbrand base of $\mathcal{P}$. $G_{\mathcal{P}}$ is a directed graph for $\mathcal{P}$, where the set of nodes N is equal to $B_{\mathcal{P}}$. For every $h_i \in \mathcal{P}$:

- if $a_j = \mathit{head}(h_i)$, then:
  - for every $a_k \in \mathit{body}^p(h_i)$: $\mathit{edge}^p(a_k, a_j) \in G_{\mathcal{P}}$,
  - for every $a_l \in \mathit{body}^n(h_i)$: $\mathit{edge}^n(a_l, a_j) \in G_{\mathcal{P}}$,
- if $\mathit{edge}^p(a_m, a_n) \in G_{\mathcal{P}}$ or $\mathit{edge}^n(a_m, a_n) \in G_{\mathcal{P}}$, then $\mathit{path}(a_m, a_n) \in G_{\mathcal{P}}$,
- if, $\mathit{edge}^p(a_m, a_n) \in G_{\mathcal{P}}$ or $\mathit{edge}^n(a_m, a_n) \in G_{\mathcal{P}}$, and $\mathit{path}(a_n, a_o) \in G_{\mathcal{P}}$, then $\mathit{path}(a_m, a_o) \in G_{\mathcal{P}}$.

$\mathit{edge}^p(a_i, a_j)$ denotes *positive edge*, while $\mathit{edge}^n(a_i, a_j)$ denotes *negative edge*, between atoms a_i and a_j.

Graph $G_{\mathcal{P}}$ for logic program $\mathcal{P} = \{a_2 \leftarrow a_1, \sim a_4 \ ; \ a_3 \leftarrow a_1\}$ from example 2.21 look as follows:

Definition 2.48. Let $\mathcal{P}$ be a logic program and $G_{\mathcal{P}}$ a directed graph for $\mathcal{P}$. We say that:

- atom a_j *depends positively* on atom a_i if there is $\mathit{path}(a_i, a_j)$ in $G_{\mathcal{P}}$ which consists only of positive edges,
- atom a_j *depends negatively* on atom a_i if there is $\mathit{path}(a_i, a_j)$ in $G_{\mathcal{P}}$ which contains at least one negative edge.

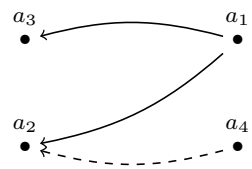

Fig. 2.1 Directed graph $G_{\mathcal{P}}$ for a logic program $\mathcal{P} = \{a_2 \leftarrow a_1, \sim a_4 \ ; \ a_3 \leftarrow a_1\}$; full arrows denote positive edges, while dashed arrows denote negative edges.

We will now define a logic program $\mathcal{P}^-$ which consists only of clauses from a logic program $\mathcal{P}$ whose head belong to the set of atoms, where for each of those atoms there is an atom that occurs negatively in $\mathcal{P}$ and depends on it. At first we define $Neg_{\mathcal{P}}$ denoting the set of all atoms that occur negatively in $\mathcal{P}$.

Definition 2.49. Let $\mathcal{P}$ be a logic program. The set of all atoms that occur negatively in $\mathcal{P}$ is defined in the following way:

$$Neg_{\mathcal{P}} =_{df} \{a_i \mid a_i \in body^n(h_j),\ h_j \in \mathcal{P}\}$$

Thereafter, by $Neg^*_{\mathcal{P}}$ we will denote the set of all atoms form logic program $\mathcal{P}$ that each atom from set $Neg_{\mathcal{P}}$ depends on.

Definition 2.50. Let $\mathcal{P}$ be a logic program and $Neg_{\mathcal{P}}$ the set of all atoms that occur negatively in $\mathcal{P}$. By $Neg^*_{\mathcal{P}}$ we define the following set:

$$Neg^*_{\mathcal{P}} =_{df} \{a_i \in B_{\mathcal{P}} \mid path(a_i, a_j) \in G_{\mathcal{P}},\ a_j \in Neg_{\mathcal{P}}\}$$

A logic program $\mathcal{P}^-$ is a set of clauses from $\mathcal{P}$ whose heads are elements of $Neg^*_{\mathcal{P}}$.

Definition 2.51. Let $\mathcal{P}$ be a logic program. We define logic program $\mathcal{P}^-$ in the following way:

$$\mathcal{P}^- =_{df} \{h_i \in \mathcal{P} \mid head(h_i) \in Neg^*_{\mathcal{P}}\}$$

Looking back at our logic program form example 2.21 for which we created a graph $G_{\mathcal{P}}$ (see Figure 2.1) we have the following:

$$\begin{aligned} \mathcal{P} &= \{a_2 \leftarrow a_1, \sim a_4 \ ; \ a_3 \leftarrow a_1\} \\ Neg_{\mathcal{P}} &= \{a_4\} \\ Neg^*_{\mathcal{P}} &= \emptyset \\ \mathcal{P}^- &= \emptyset \end{aligned} \tag{2.30}$$

Definition of acceptable logic programs requires interpretation of the negation as finite failure in classical propositional logic (CPL). In order to be able

to do so we are going to define a *completion* of a logic program $\mathcal{P}$. Firstly, we will introduce a *definition* of an atom a_i in a logic program $\mathcal{P}$, which is a set of all Horn clauses from $\mathcal{P}$ with a_i as a head.

Definition 2.52. Let a_i be an atom and $\mathcal{P}$ a logic program. The *definition* of a_i in $\mathcal{P}$ is the following set:

$$\mathsf{d}(a_i) = \{h_j \in \mathcal{P} \mid a_i = head(h_j)\}$$

We will say that Horn clauses that belong to the set $\mathsf{d}(a_i)$ *define* the atom a_i in the logic program $\mathcal{P}$. If there is an atom $a_i \in B_{\mathcal{P}}$ and $a_i \notin B^h_{\mathcal{P}}$ for some logic program $\mathcal{P}$, then we will say that a_i is *not defined* in $\mathcal{P}$.

On the basis of a definition of an atom a_i in a logic program $\mathcal{P}$ we will create a *completed definition* of an atom a_i in $\mathcal{P}$. The completed definition of a_i is an expression of the language $\mathcal{L}_{\mathsf{CPL}}$, where $\mathcal{L}_{\mathsf{CPL}}$ denotes the language of CPL with the top element $\top$, defined as usual. Similarly, the set of formulas of language $\mathcal{L}_{\mathsf{CPL}}$ is defined in a standard way.

Definition 2.53. Let a_i be an atom, $\mathcal{P}$ a logic program and $\mathsf{d}(a_i)$ a definition of a_i in $\mathcal{P}$. Let us assume that the definition of a_i in $\mathcal{P}$ is of the following form:

$$\mathsf{d}(a_i) = \{a_i \leftarrow B_1, a_i \leftarrow B_2, \ldots, a_i \leftarrow B_n\}$$

where B_j $(1 \leq j \leq n)$ is the body of j-th Horn clause from $\mathsf{d}(a_i)$. The *completed definition* of a_i denoted by $\mathsf{cd}(a_i)$ is the following expression:

$$a_i \rightarrow (B'_1 \vee B'_2 \vee \cdots \vee B'_n)$$

where B'_j $(1 \leq j \leq n)$ is obtained from B_j by replacing the commas separating literals by $\wedge$ and all occurrences of default negation $\sim$ by classical negation $\neg$. If it is the case that B_j is empty, i.e. a_i is a fact, then the top element $\top$ is inserted instead of B_j.

Let us consider the logic program $\mathcal{P}$ from example 2.21:

$$\mathcal{P} = \{a_2 \leftarrow a_1, \sim a_4 \ ; \ a_3 \leftarrow a_1\}$$

The definition and the completed definition of atom a_2 in $\mathcal{P}$ look as follows:

$$\begin{aligned} \mathsf{d}(a_2) &= \{a_2 \leftarrow a_1, \sim a_4\} \\ \mathsf{cd}(a_2) &= \{a_2 \rightarrow a_1 \wedge \neg a_4\} \end{aligned} \tag{2.31}$$

The *Clark completion* of a logic program $\mathcal{P}$ is a translated version of $\mathcal{P}$ into a CPL, which was introduced by Clark as a way of capturing the meaning of the negation by finite failure in classical propositional logic [7]. The completed version of a logic program $\mathcal{P}$ contains Horn clauses from $\mathcal{P}$ with changed negation and commas replaced by conjunctions, completed definitions of atoms defined in $\mathcal{P}$ and additional classically negated atoms for those atoms that occur in $\mathcal{P}$ but are not defined.

Definition 2.54. Let $\mathcal{P}$ be a logic program. The *Clark completion* of $\mathcal{P}$ denoted by $\mathsf{comp}(\mathcal{P})$ is a set of clauses of $\mathcal{L}_{\mathsf{CPL}}$ containing the following elements:

- clauses from $\mathcal{P}$ with default negation ($\sim$) replaced by the classical negation ($\neg$), commas in the bodies replaced by conjunction ($\wedge$) and arrow ($\leftarrow$) treated as a material implication,
- completed definitions of atoms defined in $\mathcal{P}$,
- expressions of the form: $\neg a_i$, for every atom $a_i \in B_{\mathcal{P}}$ not defined in $\mathcal{P}$.

The completed version of the logic program $\mathcal{P}$ from example 2.21 looks as follows:

$$\begin{aligned}\mathsf{comp}(\mathcal{P}) = \{&a_1 \wedge \neg a_4 \rightarrow a_2 \ ;\\ &a_1 \rightarrow a_3 \ ;\\ &a_2 \rightarrow a_1 \wedge \neg a_4 \ ;\\ &a_3 \rightarrow a_1 \ ;\\ &\neg a_1 \ ; \ \neg a_4\}\end{aligned} \tag{2.32}$$

Similarly as in the case of logic programs, $\mathsf{comp}(\mathcal{P})$ is a set of formulas from CPL, but instead of separating items by commas we will use a semicolon sign. Additionally, changed Horn clauses from logic program $\mathcal{P}$ and their completed definitions can be merged into equivalence, therefore the completion of $\mathcal{P}$ from example 2.32 can be rewritten as follows:

$$\begin{aligned}\mathsf{comp}(\mathcal{P}) = \{&a_2 \leftrightarrow a_1 \wedge \neg a_4 \ ;\\ &a_3 \leftrightarrow a_1 \ ;\\ &\neg a_1 \ ; \ \neg a_4\}\end{aligned} \tag{2.33}$$

The last technical notion used in a definition of an acceptable logic program is a *level mapping* function, which assigns to an atom a_i from the Herbrand base $B_{\mathcal{P}}$ of a logic program $\mathcal{P}$ an arbitrary natural number.

Definition 2.55. Let $\mathcal{P}$ be a logic program and $B_{\mathcal{P}}$ a Herbrand base of $\mathcal{P}$. A *level mapping* is a function $|\cdot| : B_{\mathcal{P}} \longrightarrow \mathbb{N}$. $|a_i|$ is the *level of* a_i and $|a_i| = |\sim a_i|$.

An acceptable logic program $\mathcal{P}$ is a program for which we can find an interpretation I (not necessarily a Herbrand one) that is a model for $\mathcal{P}$ and $\mathsf{comp}(\mathcal{P}^-)$ after subtraction from I all atoms that are not elements of $Neg^*_{\mathcal{P}}$, and additionally we can find a level mapping $|\cdot|$ that assigns greater number to the head of a Horn clause than to the first atom (looking from the left to right, i.e. from arrow to the end of the body) from the body of that Horn clause that is not a logical consequence of interpretation I, for every Horn clause from $\mathcal{P}$.

Definition 2.56. Let $\mathcal{P}$ be a logic program, $|\cdot|$ a level mapping for $\mathcal{P}$, and I a model for $\mathcal{P}$ whose restriction to the atoms in $Neg^*_{\mathcal{P}}$ is a model for the

completion of $\mathcal{P}^-$. Logic program $\mathcal{P}$ is called *acceptable* w.r.t. $|\cdot|$ and I if, for every clause $a_p \leftarrow l_{t_1}, \ldots, l_{r_n}$ in $\mathcal{P}$, the following implication holds for $1 \leq i \leq n$:

$$\text{if } I \models \bigwedge_{j=1}^{i-1} l_{q_j} \text{ then } |a_p| > |l_{s_i}|$$

In other words, $\mathcal{P}$ is *acceptable* w.r.t. $|\cdot|$ and I if

$$|a_p| > |l_{q_i}| \text{ for } i \in [1, \overline{n}]$$

where

$$\overline{n} = \min(\{n\} \cup \{i \in [1, n] \mid I \not\models l_{q_i}\})$$

$\mathcal{P}$ is called *acceptable* if it is acceptable w.r.t. some level mapping and a model for $\mathcal{P}$.

The notion of acceptability of a logic program was defined as a tool for checking if a program written in Prolog can successfully execute the computation. Prolog execution rule checks atoms from the body of a given Horn clause in the order from left to right (i.e. from arrow to the end of the body). Therefore the placement of atoms in the body of Horn clauses in logic program $\mathcal{P}$ is important if the acceptability property is tested with the use of the definition 2.56. For example, from the two logic programs

$$\mathcal{P}_1 = \{a_1 \leftarrow \sim a_1, a_2\}$$
$$\mathcal{P}_2 = \{a_1 \leftarrow a_2, \sim a_1\}$$

only the second one, i.e. $\mathcal{P}_2$, is acceptable. As we will see in next chapter, neural network obtained from a logic program computes the head of a given Horn clause by taking into account simultaneously all atoms from the body of that clause. For that reason there would be no difference between neural networks obtained from logic program $\mathcal{P}_1$ and $\mathcal{P}_2$. We will modify the definition of acceptable logic program in such way that the ordering of atoms in bodies of Horn clauses is no longer significant, as in [14, p. 251].

Definition 2.57. Let $\mathcal{P}$ be a logic program. $\mathcal{P}$ is acceptable if it is acceptable w.r.t. a definition 2.56 for some ordering of the atoms in the bodies of the Horn clauses from $\mathcal{P}$.

Finally, the theorem which determines that for every acceptable logic program the immediate consequence operator has a unique fixpoint can be proven.

Theorem 2.4. *Let $\mathcal{P}$ be an acceptable logic program. The following can be proven [3]:*

- *$T_{\mathcal{P}} \uparrow \omega$ is the unique fixpoint of $T_{\mathcal{P}}$,*
- *$T_{\mathcal{P}} \uparrow \omega = T_{\mathcal{P}} \downarrow \omega$.*

2.5 Chapter summary

The Chapter begins with the description of the mathematical background of propositional account on logic programs. After a brief introduction of some of the auxiliary notions that were used in the later part of the Chapter the definite logic programs were introduced with all the formal background. The goal of this Section (see p. 15) was to introduce Theorem 2.1, which states that that a Horn clause h_i is a logical consequence of a logic program $\mathcal{P}$ iff its head belongs to the least Herbrand model for the modified logic program $\mathcal{P}$ w.r.t. the Horn clause h_i. In order to prove Theorem 2.1 some auxiliary notion were introduced. The first one was chain mapping (Definition 2.22 on p. 19) which is used to determine if there is a *connection* between a set of atoms and an atom by means of Horn clauses from a given logic program. On this basis Proposition 2.7 was proved, which states that there is a strict correspondence between the result of the chain mapping with arguments: a logic program, the head of a Horn clause, a set of atoms that form the body of that Horn clause, and the ability of finding a model for the logic program, which in the same time is not a model for the Horn clause. The modification of a logic program w.r.t. a Horn clause (Definition 2.32 on p. 30) was another such notion. A logic program modified w.r.t. a Horn clause is the logic program augmented with atoms from the body of the Horn clause in the form of facts. In Corollary 2.7 is was proved that chain mapping with arguments: a logic program, the body of a Horn clause, the head of the Horn clause, returns positive answer iff every model for the logic program modified w.r.t. the Horn clause contains the head of the Horn clause. Finally, on the basis of Theorem 2.1 an abductive goal in the form of a Horn clause was defined (Definition 2.33 on p. 2.33).

The following Section (see p. 35) characterises the formalisation of the normal logic programs, i.e. logic programs which contain Horn clauses with negation allowed in their bodies. Similarly as in the case of definite logic programs, the goal here was to define logical consequence of the form of a Horn clause for a logic program in terms of the immediate consequence operator. However, normal logic programs do not have to have the least Herbrand model but instead they can have many different minimal models and by means of the immediate consequence operator not all of those minimal models can be found. Therefore, Theorem 2.3 proved at the end of the Section 2.4 (see p. 43), which states that, if the head of a Horn clause h_i belongs to the least Herbrand model of modified positive version of a logic program $\mathcal{P}$ w.r.t. the positive version of the clause h_i, then the clause h_i is a logical consequence of the logic program $\mathcal{P}$, grants us that the method of establishing if a given Horn clause is a logical consequence of a logic program based on the immediate consequence operator is sound but not complete. The proof of Theorem 2.3 is based on the Theorem 2.1 from the previous Section. The reason which allows to use theorem for definite in case of normal logic programs is that the normal logic program is transformed into its positive version (see

Definition 2.45 on p. 41), i.e. all negated atoms are converted into atoms with the upper index n, and hence the positive normal logic program is a definite one.

There is one more problem with the immediate consequence operator when normal logic programs are concerned. Namely, there are logic programs for which the immediate consequence operator does not have a fixpoint. In situation such as this a neural network which is a result of translation of a logic program cannot stabilise itself, i.e. the state of the output layer neurons changes every cycle. For that reason a notion of an acceptable logic program was introduced in Subsection 2.4.1. The immediate consequence operator always has a fixpoint when acceptable logic programs are concerned—regardless of the initial interpretation—which is a minimal model for that logic program.

The subsection concerning acceptable logic programs closes this chapter. Now directions for the future work will be described, which involve a third kind of logic programs called extended logic programs. An extended logic program is a set of Horn clauses of the following form:

$$l_i \leftarrow l_j, \ldots, l_k, \sim l_l, \ldots, \sim l_m$$

where l_i, l_j, l_k, l_l, l_m stand for classically understood literals, i.e. atom or its classical negation, while the meaning of $\sim$ is the same as in this work, i.e. it is understood as a negation by finite failure. Garcez et al. [14] created a translation algorithm for extended logic programs into neural networks. A neural network created by means of this algorithm computes the Answer Set Semantics [21] for extended logic programs. Generally speaking, the answer set which is a stable model [20] for a given extended logic program and which consists of atoms and classically negated atoms, gives an answer for each query: *yes*, if an atom from the query is an element of the stable model, *no*, if a negation of an atom from the query is an element of the stable model, and finally, *unknown*, if neither of these two situations occur. The semantics of stable models is briefly described in the Chapter 6 (see p. 119) devoted to the description of already existing abductive procedures and their comparison with results obtained in this work. Similarly as in the case of definite and normal logic programs, the goal here would be to create a connection between the immediate consequence operator and the method of establishing if a given Horn clause is a logical consequence of an extended logic program.

Chapter 3
Neural networks

The beginning of research on artificial neural networks can be traced back to year 1943, when McCulloch and Pitts published a paper in which they described the model of an artificial threshold neuron [37]. The next several years of development of the theory concerning artificial neural networks yielded many results amongst which there was the discovery of the property of neural networks of universal computation, i.e. the ability to perform any computation that can be made by means of a digital computer [25]. The next big breakthrough moment was when Rosenblatt and his group proposed neural networks organised in layers with feedforward connections between layers, called *perceptrons* [46]. However, neural networks alone are not that much useful. There should be also a method of constructing them in such a way that given a problem or computation to convey we are able to create a particular neural network that is able to solve the problem or to compute desired values. The backpropagation algorithm was the first such method that allowed to train multilayered neural networks, however it was "not yet the holy grail of a completely general algorithm able to teach an arbitrary computational task to a network" [25, p. 8]. The first use of the backpropagation algorithm was in 1974 by Paul Werbose, who described it in his PhD thesis [61]. It was rediscovered by Rumelhart, Hinton and Williams in 1985 [47] and published in article in 1986 [48]. Nowadays using the backpropagation algorithm is still the most common way of artificial neural networks training.

Artificial neural networks serve as models of phenomenons that are hard to describe by means of equations or are used to find good approximate solutions to complex problems that are intractable [24, p. 2]. In other words, we can find neural networks being used in almost every branch of knowledge and industry, where the domain of interest relates anyhow to the mentioned issues.

Haykin [24, pp. 2–3] lists nine major properties and capabilities of artificial neural networks that make them useful or attractive in the field of computation and problems solving: nonlinearity, input-output mapping, adaptivity, evidential response, contextual information, fault tolerance, VLSI imple-

mentability, uniformity of analysis and design, neurobiological analogy. The first three properties concern the ability of neural networks to learn solving complex problems that can have nonlinear nature by establishing the input-output relation from the data. In addition, neural networks can overcome any noise that the data can have and that makes them very useful in a changing environment, where the re is need to retrain the system from time to time. Neural networks can be designed in such a way that the response for the input information provides also the confidence of the given answer, which counts for the evidential response. The following two properties regard to the knowledge representation in neural networks. Since the process of training affects the whole structure of a neural network, we can say that the neural network represents itself the knowledge contained in it. For this reason neural networks are not susceptible to minor damages or errors in their structure. The VLSI short cut stands for *very-large-scale-integrated* technology, i.e. the massively parallel nature of neural networks makes them potentially very fast. Finally, the last two properties concern the universal character of the structures used in neural networks, despite different notations and implementations, and problems they are used to, what allows to easily move between different branches of the knowledge in search for the solutions.

The Chapter is organised in the following way: firstly I will describe basic technical notions used in neural networks domain along with their mathematical background in order to make the Chapter self-contained. Since neural networks that will be constructed in this work are based on the formal structures of logic programs and abductive goals, and additionally, I want to describe a detailed translation algorithm from logic programs to neural networks, it is convenient to introduce a special notation used to characterise a neural network. Those issues are covered in the second Section of this Chapter.

3.1 The background

An artificial neural network can be described as a directed graph where units of the graph are called *neurons* and connections between nodes are weighted.

The model of a typical artificial neuron consists of the following three elements [24, p. 10–11]:

1. A set of *synapses* (*connecting links*). For a given neuron k, every synapse delivers an input signal x_i magnified by the *weight* w_{ki}.
2. An *adder*, which sums all the weighted input signals.
3. An *activation function*. For a given neuron k, the output signal y_k is equal to the result of applying the activation function to the result obtained from the adder.

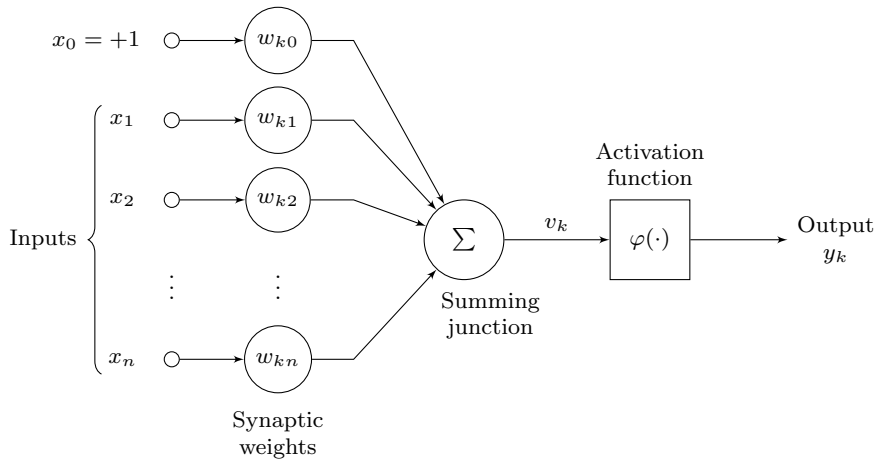

Fig. 3.1 Block diagram of a model of a neuron. [24, p. 12]

The block diagram of a model of a neuron is shown in the Figure 3.1. Input signals are denoted as x_0, x_1, ..., x_n, where $x_0 = +1$ is a fixed input signal. Together with synaptic weights for the input connections w_{k0}, w_{k1}, ..., w_{kn}, where index k refers to the neuron given in the diagram, they form the set of synapses for the neuron k. The fixed input x_0 modified by the weight w_{k0} is called the *bias* (or *threshold*) of the neuron k and can be marked by b_k:

$$b_k = x_0 w_{k0} \tag{3.1}$$

All synapses are connected with the adder or the *summing junction* as can be seen in the Figure 3.1. The adder can be described by the following equation, where v_k is the output signal from the adder called the *induced local field* or the *activation potential*:

$$v_k = \sum_{i=0}^{n} w_{ki} x_i \tag{3.2}$$

Using the special notation for the bias from Equation 3.1, we can rewrite Equation 3.2 for the activation potential as two equations:

$$u_k = \sum_{i=1}^{n} w_{ki} x_i \tag{3.3}$$

$$v_k = b_k + u_k \tag{3.4}$$

where u_k is called the *linear combiner* [24, p. 11]. Given that the activation function for the neuron k is $\varphi(\cdot)$, we can describe the output state of the neuron by the following equation:

$$y_k = \varphi(v_k) \tag{3.5}$$

There are three basic activation functions (diagrams for activation functions are shown in the Figure 3.2):

1. *Linear* activation function:

$$\varphi(v_k) = v_k \tag{3.6}$$

2. *Threshold* or *non-linear* activation function:

$$\varphi(v_k) = \begin{cases} 1 & \text{if } v_k \geq 0 \\ 0 & \text{if } v_k < 0 \end{cases} \tag{3.7}$$

3. *Sigmoid* or *semi-linear* activation function:

$$\varphi(v_k) = \frac{1}{1 + \exp(-\beta v_k)} \tag{3.8}$$

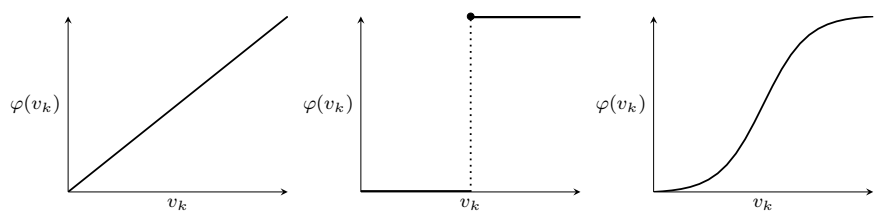

Fig. 3.2 Activation functions; from left to right: linear, threshold and sigmoid activation function.

The values of threshold and the sigmoid activation functions defined as in the equations 3.7 and 3.8 range from 0 to +1. It could be the case that the desired range for those functions is from −1 to 1. In such situation the threshold activation function is defined as follows:

$$\varphi(v_k) = \begin{cases} 1 & \text{if } v > 0 \\ 0 & \text{if } v = 0 \\ -1 & \text{if } v < 0 \end{cases} \tag{3.9}$$

and the sigmoid activation function is defined in the following way:

$$\varphi(v_k) = \frac{2}{1 + \exp(-\beta v_k)} - 1 \tag{3.10}$$

In order to be able to display neural networks in a clear and transparent way we will use an *architectural graph* to represent neural networks [24, p. 17]. Each neuron is called a *computation node* and is represented by a single

shaded circle, as can be seen in the Figure 3.3, where a single neuron with n source nodes (the x_0 input signal is fixed and represents the bias) is depicted. The block diagram from Figure 3.1 and the architectural graph from the Figure 3.3 represent the same neuron unit.

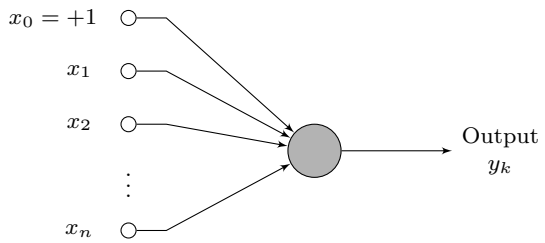

Fig. 3.3 Architectural graph of a neuron.

Neurons in neural networks can be organised in many different ways, but here we will only deal with *layered* neural networks, where neurons are organised in layers. The translation algorithm described in the next Section (see page 65) produce only three layered neural networks, which are composed of the input, hidden and output layer of neurons. These neural networks are of a *feedforward* type (i.e. the input signal is propagated only in one direction—to the output layer) and are clearly not *fully connected*, since in a fully connected network all neurons from the input layer are connected with all the neurons from the hidden layer, which in turn are connected with all the neurons from the output layer. Additionally, neural networks that are of our interest are *partially recurrent* [14], where recurrent connections are made from output neurons to particular input neurons. In the Figure 3.4 a neural network is depicted which is three-layered, feedforward and partially recurrent.

3.2 The language

Now the special notation will be introduced which allows to precisely describe both algorithms in the next Chapter (see page 66), i.e. the algorithm that translates logic programs into neural networks and the algorithm that translates neural networks into logic programs.

According to the block model from the Figure 3.1 and equations 3.1–3.3 to describe a neuron k we need to specify the following elements:

- the set of input connections $\{x_i\}_{i=1}^n$,
- the set of weights for the input connections $\{w_{ki}\}_{i=1}^n$,
- the value of the bias b_k,

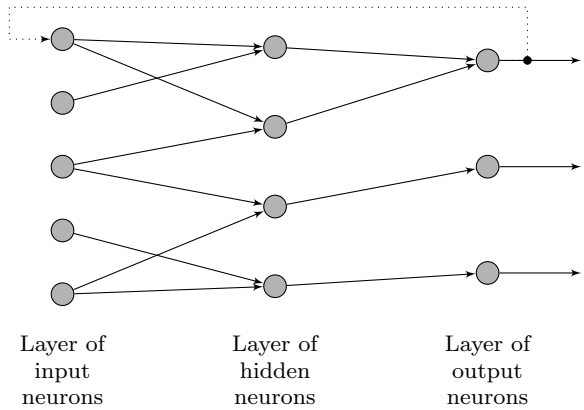

Fig. 3.4 Partially connected, feedforward neural network with three layers of neurons: input, hidden and output layer, and one recurrent connection.

- the activation function $\varphi(\cdot)$.

The first two items, i.e. the information about the input connections along with their weights will be stored in a separate set. Additionally, each neuron should be equipped with the information about the association of the neuron with the atom from the logic program (*label*) and the information about the interpretation of the activation of the neuron in terms of the *truth*/*false* values of the valuation for the atom associated with that neuron ($A_{\min}$).

Definition 3.1. A *neuron* denoted by $\mathfrak{n}_i$ is the following tuple:

$$\langle \varphi(\cdot), b_{\mathfrak{n}_i}, \mathit{label}, \mathit{amin} \rangle$$

We can distinguish four kinds of neurons in our neural networks:

- input layer neurons (denoted by $\mathfrak{i}_i$),
- hidden layer neurons (denoted by $\mathfrak{h}_i$),
- output layer neurons (denoted by $\mathfrak{o}_i$),
- additional recursive neurons (denoted by $\mathfrak{r}_i$).

Neurons from given layer are defined as specific instances of the general neuron defined in 3.1. The fourth kind of neurons was not mentioned before, because it relates to the problem that arises when we translate a logic program along with an abductive goal into a neural network that is described in details at the end of the Subsection 4.1.1.2 (see p. 75). What is important to note here is that the last kind of neurons does not influence the way the neural network works.

Since we are using two activation functions for neurons from the input, the hidden and the output layer in our neural networks, and one for the additional recursive neurons, we will denote them by fixed symbols for the rest of the work:

- $g(\cdot)$ — denoting the identity function:

$$g(x) = x \tag{3.11}$$

- $h(\cdot)$ — denoting the sigmoid function:

$$h(x) = \frac{2}{1 + \exp(-\beta x)} - 1 \tag{3.12}$$

- $k(\cdot)$ — denoting the activation function for the additional recursive neurons:

$$k(x) = \begin{cases} A_{\min} + A^a_{\min} & \text{if } x \leq A_{\min} + A^a_{\min} \\ x & \text{if } x \in (A_{\min} + A^a_{\min}; 1) \\ 1 & \text{if } x \geq 1 \end{cases} \tag{3.13}$$

Now we can define neurons from the input, the hidden and the output layer on the basis of the general definition of a neuron. Since neurons from the input and the output layer are associated with atoms from the logic program $\mathcal{P}$ or abductive goal $\mathcal{G}$ we recall the notion of a Herbrand base, i.e. the set of all atoms occurring in a logic program (denoted by $B_{\mathcal{P}}$) or in an abductive goal (denoted by $B_{\mathcal{G}}$). Neurons from the hidden layer will be labelled with numbers from indexes of Horn clauses form the logic program $\mathcal{P}$ that they are associated with.

Definition 3.2. An *input layer neuron* denoted by $\mathfrak{i}_i$ is a neuron, where:

- $\varphi(\cdot) = g(\cdot)$,
- $b_{\mathfrak{i}_i} = 0$,
- $label \in B_{\mathcal{P}} \cup B_{\mathcal{G}}$,
- $amin = A_{\min}$.

Definition 3.3. A *hidden layer neuron* denoted by $\mathfrak{h}_i$ is a neuron, where:

- $\varphi(\cdot) = h(\cdot)$,
- $b_{\mathfrak{h}_i}$ is calculated by the translation algorithm,
- $label \in \{h_j\}_{j=1}^n$; where $\{h_j\}_{j=1}^n \in \mathcal{P}$,
- $amin = A_{\min}$.

Definition 3.4. An *output layer neuron* denoted by $\mathfrak{o}_i$ is a neuron, where:

- $\varphi(\cdot) = h(\cdot)$,
- $b_{\mathfrak{o}_i}$ is calculated by the translation algorithm,
- $label \in B_{\mathcal{P}} \cup B_{\mathcal{G}}$,
- $amin = A_{\min}$.

Now we will define two special neurons which are reserved for the translation of facts: a *truth* neuron in the input layer and its hidden layer counterpart. There is only one truth neuron and its hidden layer counterpart, therefore, they will be represented by reserved symbols: $\mathfrak{t}$ and $\mathfrak{h}_{\mathfrak{t}}$ respectively.

Definition 3.5. The *truth neuron* denoted by $\mathfrak{t}$ is an input layer neuron, where:

- $\varphi(\cdot) = 1$,
- $label = t$.

Definition 3.6. The *hidden layer truth neuron* denoted by $\mathfrak{h}_{\mathfrak{t}}$ is a hidden layer neuron, where:

- $\varphi(\cdot) = 1$,
- $b_{\mathfrak{h}_{\mathfrak{t}}} = 0$,
- $label = ht$.

In order to be able to introduce new knowledge into the neural network trained by means of the Backpropagation algorithm, we will add additional hidden layer neurons. This step will allow to obtain new Horn clauses after the translation of the neural network into a logic program. Additional hidden layer neurons have label *ha* with numeration in the lower index that is separate from the numeration of hidden layer neurons and the bias is not calculated but handed over as one of the parameters of the translation algorithm—the issue is described in details in Subsection 4.1.1.2 in the next Chapter.

Definition 3.7. An *additional hidden layer neuron* is a hidden layer neuron $\mathfrak{h}_i$, where:

- $b_{\mathfrak{h}_i} = b_a$,
- $label = ha_n$.

The last kind of neurons that are used in neural networks in this work concerns units that serve as intermediary between the output and the input layer neurons. They are concerned with the recursive connections problem that is described in details in Subsection 4.1.1.2 in the next Chapter. The activation function of those neurons is the function $k(x)$ described in Equation 3.13, the bias is equal to 0 and labels are *oi* numerated with the lower index. Sice those neurons do not tke part in the translation of the neural network into a logic program, there is no need for them to have $A_{\min}$ value, hence the last element of the tuple that defines such neurons is set to 0.

Definition 3.8. An *additional recursive neuron* is a neuron $\mathfrak{r}_i$, where:

- $\varphi(\cdot) = k(x)$,
- $b_{\mathfrak{r}_i} = 0$,
- $label = oi_n$,
- $amin = 0$.

On this basis we can define sets with neurons for a neural network from a given layer: $\mathcal{N}_I$, $\mathcal{N}_H$, $\mathcal{N}_O$ for input, hidden and output layer neurons, respectively, and two sets of additional neurons: $\mathcal{N}_{AH}$ and $\mathcal{N}_{AR}$ for additional hidden and additional recursive neurons. Combining those sets together we obtain the set of all neurons $\mathcal{N}$ from a neural network.

Definition 3.9. Let $\mathcal{N}_I$, N_H and N_O be the set of all input, hidden and output neurons, respectively, and $\mathcal{N}_{AH}$ and $\mathcal{N}_{AR}$ be the set of all additional hidden layer and additional recursive neurons. By $\mathcal{N}$ we denote set of all neurons from a neural network:

$$\mathcal{N} = \mathcal{N}_I \cup \mathcal{N}_H \cup \mathcal{N}_O \cup \mathcal{N}_{AH} \cup \mathcal{N}_{AR}$$

As we have mentioned earlier, the information about connections between neurons and their weights will be stored in two separate sets. Now we are going to define a connection between two neurons, which is a tuple containing symbols of those neurons.

Definition 3.10. Let $\mathfrak{n}_i$ and $\mathfrak{n}_j$ be neurons. A *connection* between those neurons is the following pair:

$$c_{\mathfrak{n}_i \mathfrak{n}_j} =_{df} \langle \mathfrak{n}_i, \mathfrak{n}_j \rangle$$

The connection $c_{\mathfrak{n}_i \mathfrak{n}_j}$ states that the output signal from the neuron $\mathfrak{n}_i$ is an input signal for neuron $\mathfrak{n}_j$.

We can define the following sets of connections:

- connections from input to hidden layer neurons,
- connections from hidden layer to output neurons,
- recursive connections,
- additional connections.

Definition 3.11. Let $c_{\mathfrak{i}_i \mathfrak{h}_j}, \ldots, c_{\mathfrak{i}_k \mathfrak{h}_l}$ be all the connections from an input to a hidden layer neuron in a given neural network. By $\mathcal{C}_{i \to h}$ we denote the set of all connections from input to hidden layer neurons in the neural network:

$$\mathcal{C}_{i \to h} = \{c_{\mathfrak{i}_i \mathfrak{h}_j}, \ldots, c_{\mathfrak{i}_l \mathfrak{h}_l}\}$$

Definition 3.12. Let $c_{\mathfrak{h}_i \mathfrak{o}_j}, \ldots, c_{\mathfrak{h}_k \mathfrak{o}_l}$ be all the connections from a hidden to an output layer neuron in a given neural network. By $\mathcal{C}_{h \to o}$ we denote the set of all connections from hidden to output layer neurons in the neural network:

$$\mathcal{C}_{h \to o} = \{c_{\mathfrak{h}_i \mathfrak{o}_j}, \ldots, c_{\mathfrak{h}_k \mathfrak{o}_l}\}$$

Recursive connections are established in a neural network only between neurons from the output and input layer which have the same *label*.

Definition 3.13. Let $\mathfrak{o}_i$ and $\mathfrak{i}_j$ be output and input neurons in a given neural network, respectively. By $\mathcal{C}_r$ we denote the set of recursive connections from output to input neurons in the neural network:

$$\mathcal{C}_r = \{c_{\mathfrak{o}_i \mathfrak{i}_j} \mid \mathfrak{o}_i(label) = \mathfrak{i}_j(label)\}$$

In order to be able train neural network a set of additional connections will be established.

Definition 3.14. Let $\mathfrak{n}_i$, $\mathfrak{n}_j$, ..., $\mathfrak{n}_k$, $\mathfrak{n}_l$ be neurons from a given neural network. By $\mathcal{C}_a$ we denote set of the additional connections in the neural network:

$$\mathcal{C}_a = \{c_{\mathfrak{n}_i\mathfrak{n}_j}, \ldots, c_{\mathfrak{n}_k\mathfrak{n}_l}\}$$

The sum of four above defined sets is the set of all connection in the neural network.

Definition 3.15. Let $\mathcal{C}_{i\to h}$, $\mathcal{C}_{h\to o}$ and $\mathcal{C}_r$ and $\mathcal{C}_a$ be the set of input to hidden, hidden to output, recursive and additional connections in a given neural network, respectively. Then the set of all connections in the network, denoted by $\mathcal{C}$, is the following:

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

Now we can define a function which returns the weight for every connection. The weight for recursive connections is always the same and is equal to 1. Similarly, the weight of the additional connections is established as a value from the range $[-r, r]$ with the exception of 0, where r is a parameter of the neural network which is a real number close to 0. Additional connections cannot have weight equal to 0 because of the symmetry problem [14, p. 59]. The rest of the connections receive weight W calculated by the translation algorithm from the next Section.

Definition 3.16. Let $c_{\mathfrak{n}_i\mathfrak{n}_j}$ be a connection between neurons $\mathfrak{n}_i$ and $\mathfrak{n}_j$. The function $\mathsf{w} : \mathcal{C} \to \mathbb{R}$ returns a real number for a given connection in the following way:

$$\mathsf{w}(c_{\mathfrak{n}_i\mathfrak{n}_j}) = \begin{cases} 1 & \text{if } c_{\mathfrak{n}_i\mathfrak{n}_j} \in \mathcal{C}_r \\ R & \text{if } c_{\mathfrak{n}_i\mathfrak{n}_j} \in \mathcal{C}_a \\ W & \text{otherwise} \end{cases}$$

where $R \in [-r, r] \setminus \{0\}$.

Definition 3.17. Let $c_{\mathfrak{n}_i\mathfrak{n}_j}$ be a connection between neurons $\mathfrak{n}_i$ and $\mathfrak{n}_j$. A *weighted of connection* between neuron $\mathfrak{n}_i$ and $\mathfrak{n}_j$ is defined in the following way:

$$w_{\mathfrak{n}_i\mathfrak{n}_j} = \langle c_{\mathfrak{n}_i\mathfrak{n}_j}, \mathsf{w}(c_{\mathfrak{n}_i\mathfrak{n}_j})\rangle$$

Combining the set of all connections with the function that returns the weight of a given connection we obtain the set of all connections with their weights. The set $\mathcal{W}$ contains pairs, where the first element is a connection from the set of all connections $\mathcal{C}$ and the second element is the weight of the connection.

Definition 3.18. Let $w_{\mathfrak{n}_i\mathfrak{n}_j}$ be a weighted connection between neurons $\mathfrak{n}_i$ and $\mathfrak{n}_j$. The set of all weighted connections is denoted by $\mathcal{W}$:

$$\mathcal{W} = \{w_{\mathfrak{n}_i\mathfrak{n}_j} = \langle c_{\mathfrak{n}_i\mathfrak{n}_j}, \mathsf{w}(c_{\mathfrak{n}_i\mathfrak{n}_j})\rangle \mid c_{\mathfrak{n}_i\mathfrak{n}_j} \in \mathcal{C}\}$$

Bringing all pieces together we obtain a neural network which can be defined as a pair with the set of all neurons and the set of all weighted connections.

Definition 3.19. Neural network denoted by $\mathfrak{N}$ is a pair:

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

Let us consider the following example of a neural network shown in the Figure 3.5. The neural network contains the following neurons:

- four input neurons $\mathcal{N}_I = \{\mathfrak{i}_1, \mathfrak{i}_2, \mathfrak{i}_3, \mathfrak{t}\}$:

$$\begin{aligned} \mathfrak{i}_1 &= \langle g(\cdot), 0, a_1, A_{\min} \rangle \\ \mathfrak{i}_2 &= \langle g(\cdot), 0, a_4, A_{\min} \rangle \\ \mathfrak{i}_3 &= \langle g(\cdot), 0, a_3, A_{\min} \rangle \\ \mathfrak{t} &= \langle 1, 0, t, A_{\min} \rangle \end{aligned}$$

- three hidden layer neurons $\mathcal{N}_H = \{\mathfrak{h}_1, \mathfrak{h}_2, \mathfrak{h}_\mathfrak{t}\}$:

$$\begin{aligned} \mathfrak{h}_1 &= \langle h(\cdot), b_{\mathfrak{h}_1}, h_1, A_{\min} \rangle \\ \mathfrak{h}_2 &= \langle h(\cdot), b_{\mathfrak{h}_2}, h_2, A_{\min} \rangle \\ \mathfrak{h}_\mathfrak{t} &= \langle 1, 0, ht, A_{\min} \rangle \end{aligned}$$

- three output layer neurons $\mathcal{N}_O = \{\mathfrak{o}_1, \mathfrak{o}_2, \mathfrak{o}_3\}$:

$$\begin{aligned} \mathfrak{o}_1 &= \langle h(\cdot), b_{\mathfrak{o}_1}, a_2, A_{\min} \rangle \\ \mathfrak{o}_2 &= \langle h(\cdot), b_{\mathfrak{o}_2}, a_1, A_{\min} \rangle \\ \mathfrak{o}_3 &= \langle h(\cdot), b_{\mathfrak{o}_3}, a_5, A_{\min} \rangle \end{aligned}$$

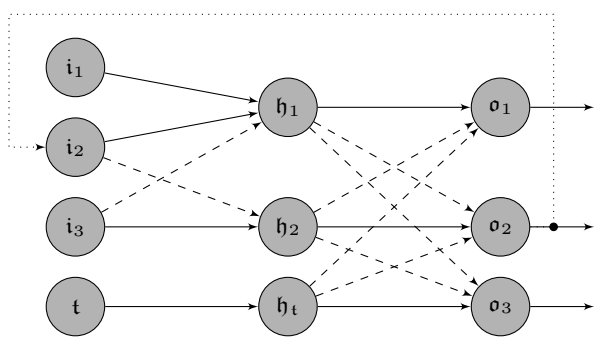

Fig. 3.5 Example of a neural network with one recursive connection (dotted line) and eight additional connections (dashed lines).

The set of weighted connections looks as follows:

$$
\begin{aligned}
\mathcal{W} = \{ & \\
& \langle c_{\mathfrak{i}_1\mathfrak{h}_1}, W\rangle, \langle c_{\mathfrak{i}_2\mathfrak{h}_1}, W\rangle, \langle c_{\mathfrak{i}_3\mathfrak{h}_2}, W\rangle, \langle c_{\mathfrak{t}\mathfrak{h}_\mathfrak{t}}, W\rangle, \\
& \langle c_{\mathfrak{h}_1\mathfrak{o}_1}, W\rangle, \langle c_{\mathfrak{h}_2\mathfrak{o}_2}, W\rangle, \langle c_{\mathfrak{h}_\mathfrak{t}\mathfrak{o}_3}, W\rangle, \\
& \langle c_{\mathfrak{o}_2\mathfrak{i}_2}, 1\rangle, \\
& \langle c_{\mathfrak{i}_2\mathfrak{h}_2}, R\rangle, \langle c_{\mathfrak{i}_3\mathfrak{h}_1}, R\rangle, \langle c_{\mathfrak{h}_1\mathfrak{o}_2}, R\rangle, \langle c_{\mathfrak{h}_1\mathfrak{o}_3}, R\rangle, \\
& \langle c_{\mathfrak{h}_2\mathfrak{o}_1}, R\rangle, \langle c_{\mathfrak{h}_2\mathfrak{o}_3}, R\rangle, \langle c_{\mathfrak{h}_\mathfrak{t}\mathfrak{o}_1}, R\rangle, \langle c_{\mathfrak{h}_\mathfrak{t}\mathfrak{o}_2}, R\rangle \\
\} &
\end{aligned}
$$

where $R \in [-r, r] \setminus \{0\}$. The first line covers connections from neurons from the input to neurons from the hidden layer, the second, connections from neurons from the hidden to neurons from output layer, then there is one recursive connection and, finally, eight additional connections: two between neurons from input and hidden layer, and six between neurons from hidden and output layer.

3.3 Chapter summary

In this Chapter the basic notions for neural networks were described along with their mathematical background. Afterwards, a notation was described that is used in the algorithm that translates a logic program into a neural network. It gives us a glimpse on the used structures and encountered problems in the construction of the translation algorithm in the next Chapter. Neurons are defined in such a way that enables to encode the data about the associations with atoms and Horn clauses from a given logic program. Additionally, each neuron is equipped with information which link the state of the neuron with the *truth/false* value of the associated atom. I have sketched also a problem that concerns the recursive connections in the neural networks. Although neural networks that model logic programs are three-layered, the definition of the immediate consequence operator given at the end of the previous Chapter enforce the implementation of additional neurons between the output and the input layer, which have a unique activation function, that resolve the problem of recursive connections described in details in Subsection 4.1.1.2 (see p. 75).

Chapter 4
Abductive hypotheses generation

According to the scheme of our abductive procedure from the Figure 1.3 (see p. 8), a logic program along with an abductive goal is translated into a neural network, which in turn is trained by means of the Backpropagation algorithm and then the trained neural network is translated back into a logic program. What we already have underlined is that the translation algorithm we want to use to translate a logic program $\mathcal{P}$ into a neural network $\mathfrak{N}$ extends the translation algorithm that Garcez et al. [14] proposed. However, the general idea of the way the translation is conveyed is preserved, i.e. the resulting neural network $\mathfrak{N}$ from a given logic program $\mathcal{P}$ simulates the way the immediate consequence operator $T_{\mathcal{P}}$ for a logic program $\mathcal{P}$ maps an input interpretation in the form of a set of atoms into another set of atoms that belongs to the Herbrand base of the logic program $\mathcal{P}$. In the previous chapter we have described the formal background for logic programs and we have also proved that consequence of a logic program $\mathcal{P}$ and what follows, the abductive goal $\mathcal{G}_{\mathcal{P}}$ for that logic program, can be defined employing the immediate consequence operator $T_{\mathcal{P}}$ (theorems 2.1, 2.3 and definition 2.33). Those results will be the foundations for the extension of the Garcez's et al. translation algorithm.

The Chapter is organised as follows. The first part concerns the integration of logic programs with neural networks, i.e. the translation algorithm proposed by Garcez et al. is described in details and afterwards, the way we have extended it to fit our purpose. Next, the translation algorithm from neural networks to logic programs is defined. The second Section is devoted to neural networks training process, i.e. the description of possible construction of the training set in such a way, that after translation of the neural network back into a logic program the Horn clause that once was the abductive goal is now, in the sense of definition 2.3, a consequence of that program. Afterwards we move to the Section where the way the abducive hypotheses are generated is described.

4.1 Integration

In the next two subsections we are going to describe algorithm that translates logic programs into neural networks, denoted by $T_{\mathcal{P}\to\mathfrak{N}}$, and algorithm that does the reverse operation—creates a logic program from a given artificial neural network, denoted by $T_{\mathfrak{N}\to\mathcal{P}}$. The first subsection is in turn divided further into two parts where the first one describes in details the algorithm proposed by Garcez et al. [14] and the second one our proposed extension of the algorithm. The subsection that concerns the translation from neural networks to logic programs is based on the work of Garcez et al.

4.1.1 Translation from logic programs to neural networks

4.1.1.1 The existing algorithm

The algorithm that translates logic programs into neural networks that we use here extends the one proposed by Garcez et al. [14, p. 48–50]. The general idea behind the algorithm defined there is that for every Horn clause from a logic program there is a hidden layer neuron, which connects neurons from the input layer that have atoms from the body of that Horn clause as labels, with the neuron from the output layer that has the head of that Horn clause as a label. Definite logic programs are special cases of normal logic programs, therefore we will describe one algorithm that translates both kinds of logic programs into neural networks. Additionally, we will only deal with the case in which all connections have the same weight[1] and the negation will be represented as a connection with a negative weight.

Let us consider the following example of a logic program:

$$\mathcal{P} = \{a_2 \leftarrow a_1, \sim a_4 \ ; \ a_1 \leftarrow a_3 \ ; \ a_5 \leftarrow\} \tag{4.1}$$

The neural network corresponding to the logic program from the example 4.1 looks as it can be seen in the Figure 4.1.

Neurons $\mathfrak{i}_1$ to $\mathfrak{i}_5$ and $\mathfrak{o}_1$ to $\mathfrak{o}_5$ correspond to the atoms a_1 to a_5 (atoms indexes do not necessarily correspond to the neurons indexes), neurons $\mathfrak{h}_1$ and $\mathfrak{h}_2$ correspond to the first and the second Horn clause from the logic program $\mathcal{P}$ and finally, neurons $\mathfrak{t}$ and $\mathfrak{h}_\mathfrak{t}$ represent the absence of the body for the fact a_5. Below is the list of all neurons along with their detailed

[1] The algorithm that Garcez et al. [14, p. 48–50] proposes allows to create artificial neural network with different weights for different connections, until all of the weights exceeds some minimal value. The purpose of the simplification in the form of a one universal weight for all connections is to improve clarity of the ideas and examples (see more on p. 68).

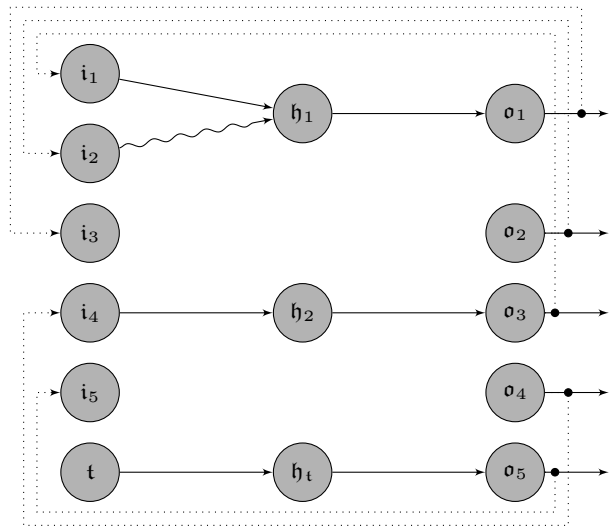

Fig. 4.1 Neural network that corresponds to the logic program from the example 4.1. Black arrows denote connection with the weight W, snake arrow denote connection with the weight $-W$ (to represent the negation in front of the atom a_4), dotted arrows denote recursive connections with the weight 1.

parameters. Connections in the neural network 4.1 are of two types: positive and negative feedforward connections with weights equal to W (arrows) and $-W$ (snake arrows), respectively; recursive connections (dotted arrows) that link neurons from the output layer with neurons from the input layer with the same label, i.e. associated with the same atom.

$$
\begin{array}{lll}
\mathfrak{i}_1 = \langle g(\cdot), 0, a_1, A_{\min}\rangle & \mathfrak{h}_1 = \langle h(\cdot), b_{\mathfrak{h}_1}, h_1, A_{\min}\rangle & \mathfrak{o}_1 = \langle h(\cdot), b_{\mathfrak{o}_1}, a_2, A_{\min}\rangle \\
\mathfrak{i}_2 = \langle g(\cdot), 0, a_4, A_{\min}\rangle & \mathfrak{h}_2 = \langle h(\cdot), b_{\mathfrak{h}_2}, h_2, A_{\min}\rangle & \mathfrak{o}_2 = \langle h(\cdot), b_{\mathfrak{o}_2}, a_4, A_{\min}\rangle \\
\mathfrak{i}_3 = \langle g(\cdot), 0, a_2, A_{\min}\rangle & \mathfrak{h}_\mathfrak{t} = \langle 1, 0, ht, A_{\min}\rangle & \mathfrak{o}_3 = \langle h(\cdot), b_{\mathfrak{o}_3}, a_1, A_{\min}\rangle \\
\mathfrak{i}_4 = \langle g(\cdot), 0, a_3, A_{\min}\rangle & & \mathfrak{o}_4 = \langle h(\cdot), b_{\mathfrak{o}_4}, a_3, A_{\min}\rangle \\
\mathfrak{i}_5 = \langle g(\cdot), 0, a_5, A_{\min}\rangle & & \mathfrak{o}_5 = \langle h(\cdot), b_{\mathfrak{o}_5}, a_5, A_{\min}\rangle \\
\mathfrak{t} = \langle 1, 0, t, A_{\min}\rangle & &
\end{array}
$$

In order for a neural network to operate as the immediate consequence operator for a logic program that the neural network was based on, weights and biases have to have specific values. We will now consider an example which will illustrate how biases and weights are calculated. For this purpose let us assume a logic program $\mathcal{P}$ which has the following Horn clause (it may contain other Horn clauses as well):

$$h_x : a_0 \leftarrow a_1, \ldots, a_k, \sim a_l, \ldots, \sim a_m \tag{4.2}$$

with body where $a_1, \ldots, a_k$ are atoms and $a_l, \ldots, a_m$ are negated atoms. The corresponding neural network structure can be seen in the Figure 4.2.

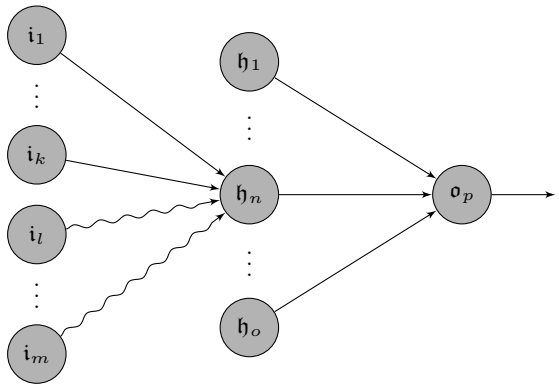

Fig. 4.2 Single Horn clause example

Input layer neurons $\mathfrak{i}_1, \ldots, \mathfrak{i}_k$ are associated with atoms $a_1, \ldots, a_k$ and $\mathfrak{i}_l, \ldots, \mathfrak{i}_m$ are associated with atoms $a_l, \ldots, a_m$, hidden layer neurons $\mathfrak{h}_1, \ldots, \mathfrak{h}_n, \ldots, \mathfrak{h}_o$ are associated with Horn clauses from logic program $\mathcal{P}$ which have the same head, and where $\mathfrak{h}_n$ represents the Horn clause 4.2, which is of our interest, and finally, output layer neuron $\mathfrak{o}_p$ is associated with the head of Horn clause 4.2, i.e. atom a_0. There are two types of connections in the Figure 4.2: arrows that stand for positive connections (weight above 0) and snake arrows that stand for negative connections (weight below 0). Negative connections are introduced to reflect the fact that some atoms in the bodies of Horn clauses may be negated, as in our case are $a_l, \ldots, a_m$.

The first value we want to calculate is the bias of the hidden layer neuron $\mathfrak{h}_n$. Before we do this we have to note three important assumptions that are made here. The first one concerns the weights of the connections, i.e. we assume that every positive connection has the same weight equal to W and every negative connection has the same weight $-W$. The second assumption is that the value $A_{\min}$ is the same for every neuron in the neural network. Finally, the value $A_{\max}$ is equal to $-A_{\min}$; as Garcez et al. write [14, p. 49], this assumptions can be made without loss of generality[2].

The bias $b_{\mathfrak{h}_n}$ for the hidden layer neuron $\mathfrak{h}_n$ is calculated as average of the minimal activation potential $v^{\min}_{\mathfrak{h}_n}$ for which the neuron should be considered to be *true* and the maximal activation potential $v^{\max}_{\mathfrak{h}_n}$ for which the neuron should be considered to be *false*:

[2] As we will see, equations we are going to describe provide only the bottom limit values for $A_{\min}$ and W and therefore we can choose values that fulfil only the highest bottom limits.

$$b_{\mathfrak{h}_n} = \frac{v_{\mathfrak{h}_n}^{\min} + v_{\mathfrak{h}_n}^{\max}}{2} \tag{4.3}$$

An interpretation I satisfies the body of Horn clause h_x 4.2 iff it contains its all positive atoms and does not contain any of its negative atoms. In other words, all positive atoms from the body of Horn clause 4.2 are mapped to *true* while all negative atoms are mapped to *false*. Therefore, the minimal activation potential $v_{\mathfrak{h}_n}^{\min}$ reflects a situation in which the following conditions are met:

- neurons $\mathsf{i}_1, \ldots, \mathsf{i}_k$, that are associated with positive atoms from the body of h_x, have the minimal activation potential that allows to consider them active, i.e. $A_{\min}$,
- neurons $\mathsf{i}_l, \ldots, \mathsf{i}_m$, that are associated with negative atoms from the body of h_x, have maximal activation potential that allows to consider them to be inactive, i.e. $-A_{\min}$.

Since the activation potential depends on the number of the positive and negative atoms in the body of a Horn clause (as can be seen in the Figure 4.2), we have to introduce two additional functions that returns the number of positive and negative atoms in the body of a Horn clause and their sum.

Definition 4.1. Let h_i be a Horn clause. Mapping $\mathsf{p} : \mathit{Hcl} \longrightarrow \mathbb{N}$ is defined as follows:

$$\mathsf{p}(h_i) = \mathsf{c}(\mathit{body}^p(h_i))$$

Definition 4.2. Let h_i be a Horn clause. Mapping $\mathsf{n} : \mathit{Hcl} \longrightarrow \mathbb{N}$ is defined as follows:

$$\mathsf{n}(h_i) = \mathsf{c}(\mathit{body}^n(h_i))$$

Definition 4.3. Let h_i be a Horn clause. The mapping $\mathsf{k} : \mathit{Hcl} \longrightarrow \mathbb{N}$ is defined as follows:

$$\mathsf{k}(h_i) = \mathsf{c}(\mathit{body}^a(h_i))$$

Now we can write down the equation for the activation potential $v_{\mathfrak{h}_n}^{\min}$:

$$\begin{aligned} v_{\mathfrak{h}_n}^{\min} &= \mathsf{p}(h_x)A_{\min}W + \mathsf{n}(h_x)(-A_{\min})(-W) = \\ &= \mathsf{p}(h_x)A_{\min}W + \mathsf{n}(h_x)A_{\min}W \end{aligned} \tag{4.4}$$

Similarly, an interpretation I does not satisfy the body of a Horn clause h_x 4.2 if it does not contain at least one positive atom or it contain at least one negative atom. However, the maximal activation potential $v_{\mathfrak{h}_n}^{\max}$ would be generated in the first case. Therefore, the activation of neurons $\mathsf{i}_1, \ldots, \mathsf{i}_{k-1}$ associated with positive atoms is equal to 1 and the activation of neuron i_k is maximal, i.e. it is equal to $-A_{\min}$, and the activation of neurons $\mathsf{i}_l, \ldots, \mathsf{i}_m$ associated with negative atoms is equal to -1:

$$\begin{aligned} v_{\mathfrak{h}_n}^{\max} &= (\mathsf{p}(h_x) - 1)W + (-A_{\min})W + \mathsf{n}(h_x)(-1)(-W) = \\ &= (\mathsf{p}(h_x) - 1)W - A_{\min}W + \mathsf{n}(h_x)W \end{aligned} \tag{4.5}$$

Combining equations 4.4 and 4.5 with equation 4.3 we obtain the bias of the hidden layer neuron for a given Horn clause:

$$b_{\mathfrak{h}_n} = \frac{(1 + A_{\min})(\mathsf{k}(h_x) - 1)}{2} W \tag{4.6}$$

The bias for the output layer neuron $\mathfrak{o}_p$ associated with the head of the clause h_x, that is atom a_0, depends on the number of Horn clauses in the logic program $\mathcal{P}$ that also have atom a_0 in their heads. As in the case of the bias for the hidden layer neuron, we will calculate minimal activation potential $v_{\mathfrak{o}_p}^{\min}$ for which activation of the neuron $\mathfrak{o}_p$ should exceed $A_{\min}$ value (in order to be considered as *true*) and the maximal activation potential $v_{\mathfrak{o}_p}\max$ for which the activation of the neuron $\mathfrak{o}_p$ should be smaller than $A_{\min}$ value (in order to be considered as *false*). The bias $b_{\mathfrak{o}_p}$ for the output layer neuron $\mathfrak{o}_p$ is the average between both activation potentials:

$$b_{\mathfrak{o}_p} = \frac{v_{\mathfrak{o}_p}^{\min} + v_{\mathfrak{o}_p}^{\max}}{2} \tag{4.7}$$

Knowing that the bias $b_{\mathfrak{o}_p}$ depends on the number of Horn clauses with the same head as in the given Horn clause, we are going to define mapping μ that returns the number of Horn clauses that have the same atom in their head as has the given Horn clause.

Definition 4.4. Let h_i be a Horn clause. The mapping $\mu : Hcl \longrightarrow \mathbb{N}$ is defined as follows:

$$\mu(h_i) = \mathsf{c}(\{h_j \mid h_j \in \mathcal{P},\ head(h_j) = head(h_i)\})$$

The minimal activation potential $v_{\mathfrak{o}_p}^{\min}$ is calculated for the situation when the interpretation satisfies only the body of a considered Horn clause h_x in such a way that the hidden layer neuron $\mathfrak{h}_n$ has activation equal to $A_{\min}$, while the bodies of the rest of the Horn clauses in logic program $\mathcal{P}$ with atom a_0 in their head are not satisfied, hence corresponding hidden layer neurons have activation equal to -1:

$$\begin{aligned} v_{\mathfrak{o}_p}^{\min} &= A_{\min} W + (\mu(h_x) - 1)(-1) W = \\ &= A_{\min} W - (\mu(h_x) - 1) W \end{aligned} \tag{4.8}$$

The maximal activation potential $v_{\mathfrak{o}_p}^{\max}$ is calculated for the situation when the interpretation does not satisfy any body of the Horn clauses from logic program $\mathcal{P}$ in such a way, that hidden layer neurons associated with those Horn clauses have activation equal to $-A_{\min}$ value:

$$v_{\mathfrak{o}_p}^{\max} = -A_{\min} \mu(h_x) W \tag{4.9}$$

Combining equations 4.8 and 4.9 with the equation 4.7 we obtain the output layer neuron bias for a given Horn clause h_x:

$$b_{\mathfrak{o}_p} = \frac{(1 + A_{\min})(1 - \mu(h_x))}{2} W \tag{4.10}$$

The minimal activation potentials for which considered hidden and output layer neurons are associated with *truth* value $v^{\min}_{\mathfrak{h}_n}$ and $v^{\min}_{\mathfrak{o}_p}$ should be greater than the maximal activation potentials $v^{\max}_{\mathfrak{h}_n}$ and $v^{\max}_{\mathfrak{o}_p}$ for those neurons when they are associated with *false* value (the usage of indexes "min" and "max" can be misleading, but we have to remember, that "min" corresponds to the active neuron, while "max" to the inactive):

$$v^{\min}_{\mathfrak{h}_n} > v^{\max}_{\mathfrak{h}_n}$$
$$v^{\min}_{\mathfrak{o}_p} > v^{\max}_{\mathfrak{o}_p}$$

From the above inequalities we obtain the following conditions for $A_{\min}$ value:

$$A_{\min} > \frac{\mathsf{k}(h_x) - 1}{\mathsf{k}(h_x) + 1} \tag{4.11}$$

$$A_{\min} > \frac{\mu(h_x) - 1}{\mu(h_x) + 1} \tag{4.12}$$

We can simplify both equations 4.11 and 4.12 into one condition on the ground of the assumption that the $A_{\min}$ value is the same for all neurons. To do so we need to generate values by means of the k and μ mapping for all Horn clauses in the logic program $\mathcal{P}$ and choose the highest value amongst them. We will now define two sets: one that contains values of k mapping for every Horn clause in $\mathcal{P}$ (denoted by $K_{\mathcal{P}}$) and the second one that contains values of μ mapping for every Horn clause in $\mathcal{P}$ (denoted by $Mu_{\mathcal{P}}$).

Definition 4.5. Let h_i be a Horn clause and $\mathcal{P}$ a logic program. The set $K_{\mathcal{P}}$ is defined as follows:

$$K_{\mathcal{P}} = \{\mathsf{k}(h_i) \mid h_i \in \mathcal{P}\}$$

Definition 4.6. Let h_i be a Horn clause and $\mathcal{P}$ a logic program. The set $Mu_{\mathcal{P}}$ is defined as follows:

$$Mu_{\mathcal{P}} = \{\mu(h_i) \mid h_i \in \mathcal{P}\}$$

Using the two above defined sets we can rewrite conditions 4.11 and 4.12 as one:

$$A_{\min} > \frac{\max(K_{\mathcal{P}} \cup Mu_{\mathcal{P}}) - 1}{\max(K_{\mathcal{P}} \cup Mu_{\mathcal{P}}) + 1} \tag{4.13}$$

Having defined biases for the hidden and output layer neurons we can now calculate conditions for weight W of the connections. Let us look back at

the Horn clause h_x 4.2 and its neural representation (Figure 4.2). The hidden layer neuron should be associated with the value *true* if the activation potential $v_{\mathfrak{h}_n}$ is greater or equal to the value generated by all neurons associated with positive atoms that are at the minimum activation that allows to consider them *true* (i.e. $A_{\min}$) plus the value generated by all neurons associated with negative atoms that are at the maximum activation that allows to consider them *false* (i.e. $-A_{\min}$) minus the calculated bias.

$$v_{\mathfrak{h}_n} \geq \mathsf{p}(h_x)A_{\min}W + \mathsf{n}(h_x)(-A_{\min})(-W) - b_{\mathfrak{h}_n}$$

When we substitute for the bias $b_{\mathfrak{h}_n}$ we obtain the following equation:

$$v_{\mathfrak{h}_n} \geq \mathsf{p}(h_x)A_{\min}W + \mathsf{n}(h_x)A_{\min}W - \frac{(1+A_{\min})(\mathsf{k}(h_x)-1)}{2}W \tag{4.14}$$

Additionally, we know that in order for the $\mathfrak{h}_n$ neuron to be considered *true* the value of the activation function should be greater or equal to $A_{\min}$. Therefore:

$$h(v_{\mathfrak{h}_n}) \geq A_{\min}$$

and further, using the equation for $h(\cdot)$ activation function we obtain:

$$v_{\mathfrak{h}_n} \geq -\frac{1}{\beta}\ln\left(\frac{1-A_{\min}}{1+A_{\min}}\right) \tag{4.15}$$

Combining inequalities 4.14 and 4.15 results in the following inequality:

$$\mathsf{p}(h_x)A_{\min}W + \mathsf{n}(h_x)A_{\min}W - \frac{(1+A_{\min})(\mathsf{k}(h_x)-1)}{2}W \geq -\frac{1}{\beta}\ln\left(\frac{1-A_{\min}}{1+A_{\min}}\right) \tag{4.16}$$

from which we obtain the condition for the weight W, given that $W > 0$:

$$W \geq -\frac{2}{\beta}\cdot\frac{\ln(1-A_{\min}) - \ln(1+A_{\min})}{\mathsf{k}(h_x)(A_{\min}-1)+A_{\min}+1} \tag{4.17}$$

Let us now consider the connections from hidden to output layer. The activation potential $v_{\mathfrak{o}_p}$ for the output neuron to be considered as *true* should be greater or equal than the value obtained when the hidden layer neuron $\mathfrak{h}_n$ is considered to be *true* with the minimal activation equal to $A_{\min}$, while the rest of the hidden layer neurons connected to the concerned output layer neuron $\mathfrak{o}_p$ has activation equal to -1.

$$v_{\mathfrak{o}_p} \geq A_{\min}W + (\mu(h_x)-1)(-1)W - b_{\mathfrak{o}_p}$$

After substitution for the bias $b_{\mathfrak{o}_p}$ we obtain the following equation:

$$v_{\mathfrak{o}_p} \geq A_{\min} W - (\mu(h_x) - 1)W - \frac{(1 + A_{\min})(1 - \mu(h_x))}{2} W \tag{4.18}$$

Similarly to the hidden layer neurons, we know that in order for the $\mathfrak{o}_p$ neuron to be considered *true* the value of the activation function should be greater or equal to $A_{\min}$:

$$h(v_{\mathfrak{o}_p}) \geq A_{\min}$$

and further, using the equation for $h(\cdot)$ activation function we obtain:

$$v_{\mathfrak{o}_p} \geq -\frac{1}{\beta} \ln \left(\frac{1 - A_{\min}}{1 + A_{\min}} \right) \tag{4.19}$$

Combining both inequalities 4.18 and 4.19 gives us the following inequality:

$$A_{\min} W - (\mu(h_x) - 1)W - \frac{(1 + A_{\min})(1 - \mu(h_x))}{2} W \geq - \frac{1}{\beta} \ln \left(\frac{1 - A_{\min}}{1 + A_{\min}} \right) \tag{4.20}$$

from which we obtain the condition for the weight W, given that $W > 0$:

$$W \geq -\frac{2}{\beta} \cdot \frac{\ln(1 - A_{\min}) - \ln(1 + A_{\min})}{\mu(h_x)(A_{\min} - 1) + A_{\min} + 1} \tag{4.21}$$

Similarly as in the case of the $A_{\min}$ value, we can unify conditions 4.17 and 4.21 under assumption, that every weight of the connection has the same value (where negative connections have the same value also, but negative):

$$W \geq \frac{2}{\beta} \cdot \frac{\ln(1 + A_{\min}) - \ln(1 - A_{\min})}{\max(K_{\mathcal{P}} \cup Mu_{\mathcal{P}})(A_{\min} - 1) + A_{\min} + 1} \tag{4.22}$$

Let us go back to our exemplary logic program $\mathcal{P}$ from 4.1 and the neural network 4.1 that is associated with it. We have three Horn clauses in $\mathcal{P}$:

$$\begin{array}{lll} h_1 : a_2 \leftarrow a_1, \sim a_4 & \mathsf{k}(h_1) = 2 & \mu(h_1) = 1 \\ h_2 : a_1 \leftarrow a_3 & \mathsf{k}(h_2) = 1 & \mu(h_2) = 1 \\ h_3 : a_5 \leftarrow & \mathsf{k}(h_3) = 0 & \mu(h_3) = 1 \end{array}$$

Applying equations 4.13 and 4.22 to logic program $\mathcal{P}$ from our example 4.1 we obtain the following results:

$$\begin{aligned} K_{\mathcal{P}} &= \{2, 1, 0\} \\ Mu_{\mathcal{P}} &= \{1\} \\ \max(K_{\mathcal{P}} \cup Mu_{\mathcal{P}}) &= 2 \end{aligned} \tag{4.23}$$

and the condition for $A_{\min}$ value:

$$A_{\min} > \frac{1}{3} \tag{4.24}$$

Let us assume $A_{\min} = \frac{1}{2}$. Let us further assume, that β in the activation function $h(\cdot)$ is equal to 1. The condition for weight W looks as follows:

$$W \geq 4.396 \tag{4.25}$$

Neurons $\mathfrak{h}_1$ and $\mathfrak{h}_2$ are associated with Horn clauses h_1 and h_2, respectively. Let us assume $W = 4.5$. We can now calculate biases for those neurons and neurons from the output layer by means of equations 4.3 and 4.7, respectively.

$$\begin{aligned} b_{\mathfrak{h}_1} &= \frac{27}{8} & b_{\mathfrak{o}_1} &= 0 \\ b_{\mathfrak{h}_2} &= 0 & b_{\mathfrak{o}_2} &= 0 \\ b_{\mathfrak{h}_3} &= -\frac{27}{8} & b_{\mathfrak{o}_3} &= 0 \\ & & b_{\mathfrak{o}_4} &= 0 \\ & & b_{\mathfrak{o}_5} &= 0 \end{aligned}$$

Given $A_{\min}$ value, weight W and calculated biases for hidden and output layer neurons, we have now complete information about the neural network from Figure 4.1 that is created on the basis of the logic program 4.1.

We will briefly sum up the translation algorithm from logic programs to neural networks that Garcez et al. [14] propose:

1. Create neural network with hidden neurons associated with Horn clauses from logic program $\mathcal{P}$, input and output neurons corresponding with atoms from Hebrand base $B_{\mathcal{P}}$ for $\mathcal{P}$ and additional *truth* neuron in the input layer, if $\mathcal{P}$ contains facts.
2. Connect input and hidden layer neurons, and then hidden and output layer neurons in a way that neurons from the input layers associated wit the atoms from the body of a given Horn clause are connected through a hidden layer neuron associated with that Horn clause with the output layer neuron that is associate with the head of that Horn clause.
3. Connect facts to the *truth* neuron via dedicated hidden layer neuron.
4. Add recursive connections from output to input layer neurons with the same *label*, i.e. those that are associated with the same atom form $B_{\mathcal{P}}$.
5. Calculate k and μ values for every Horn clause in logic program $\mathcal{P}$.
6. Calculate the threshold value for $A_{\min}$ and W. Establish $A_{\min}$ and W values.
7. Give every connection weight W (or $-W$, if the connection runs from input to hidden layer neuron and represents the negated atom).
8. Calculate biases for hidden and output layer neurons.

Following these steps we obtain a neural networks which works in the same way as the immediate consequence operator $T_{\mathcal{P}}$ works for the logic program $\mathcal{P}$. Thus we are able to prove the following theorem:

Theorem 4.1. *For each propositional normal logic program $\mathcal{P}$, there exists a feedforward artificial neural network $\mathfrak{N}$ with exactly one hidden layer and semi-linear neurons such that $\mathfrak{N}$ computes $T_{\mathcal{P}}$.*

The detailed proof of the Theorem 4.1 can be found in Garcez et al. [14, p. 52–55]. The proof is conveyed in a form of a detailed analysis of a general case of a Horn clause, similarly as it is given in the example 4.1, and the calculations that follow from it.

4.1.1.2 The extension

Up to this point we have described in details how the translation algorithm from logic programs to artificial neural networks that Garcez et al. [14] proposed works. Now we are going to show how a logic program along with an abductive goal can be translated into a neural network that is ready for training.

Let us recall the logic program $\mathcal{P}$ from example 4.1:

$$\mathcal{P} = \{a_2 \leftarrow a_1, \sim a_4 \ ; \ a_1 \leftarrow a_3 \ ; \ a_5 \leftarrow\}$$

The Horn clause:

$$h_g : \ a_2 \leftarrow a_3 \tag{4.26}$$

is an abductive goal $\mathcal{G}_{\mathcal{P}}$ for $\mathcal{P}$, because $head(h_g) \notin M_{\mathcal{P}_p^{\mathsf{dn}(h_g)}}$, where:

$$\begin{aligned} \mathcal{P}_p^{\mathsf{dn}(h_g)} = \{ \quad & a_2 \leftarrow a_1, a_4^n \\ & a_1 \leftarrow a_3 \\ & a_5 \leftarrow \\ & a_3^h \leftarrow \quad \} \end{aligned} \tag{4.27}$$

We want $head(h_g)$ to be an element of the $M_{\mathcal{P}_p^{\mathsf{dn}(h_g)}}$, therefore at first we build a neural network by means of the translation algorithm from Garcez et al. [14], that was described in the previous subsection, for the $\mathcal{P}_p^{\mathsf{dn}(h_g)}$ logic program. The results are shown in the Figure 4.3—there are two differences, though. The first one is that there was added a "black rectangle", which is visible in the Figure, and the second one is that there are more output layer neurons than input layer neurons, i.e. there is no input layer neuron for atom a_3^h. Those differences will be discussed in the later part of the subsection. The association between atoms from logic program $\mathcal{P}_p^{\mathsf{dn}(h_g)}$ and the neural network from the Figure 4.3 looks as shown below (the '$\leftrightsquigarrow$' sign denotes

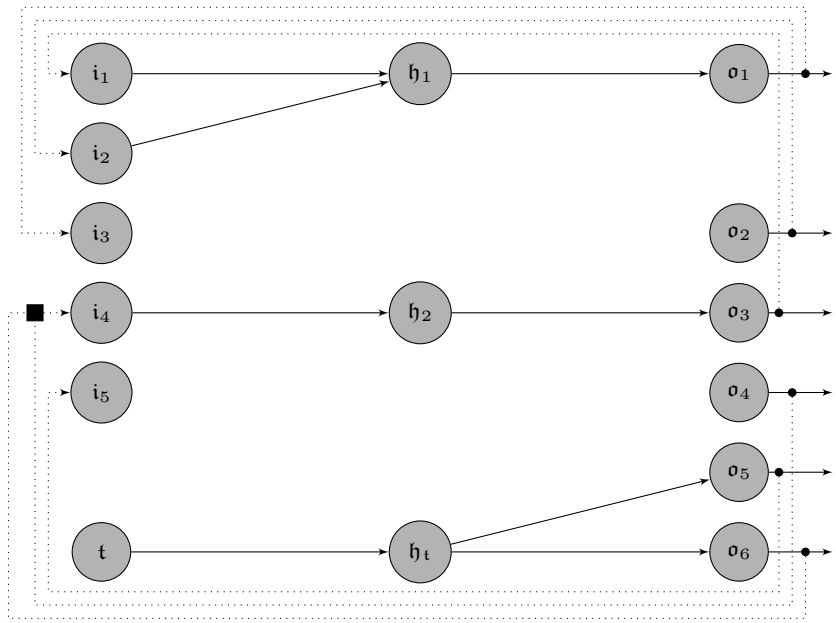

Fig. 4.3 Neural network that corresponds to the logic program from the example 4.27 created by means of the Garcez's et al. [14] algorithm. Arrows denote connections with the weight W, dotted arrows denote recursive connections with the weight 1 and the meaning of the black rectangle is discussed on p. 85.

association between given neuron and atom or Horn clause). Neurons $\mathfrak{t}$ and $\mathfrak{h}_\mathfrak{t}$ have been omitted, because they serve only to generate "truth signal" and for this reason they are not associated with any of the atoms from logic program $\mathcal{P}_p^{\mathsf{dn}(h_g)}$.

$\mathfrak{i}_1 \leftrightsquigarrow a_1$	$\mathfrak{h}_1 \leftrightsquigarrow h_1$	$\mathfrak{o}_1 \leftrightsquigarrow a_2$
$\mathfrak{i}_2 \leftrightsquigarrow a_4^n$	$\mathfrak{h}_2 \leftrightsquigarrow h_2$	$\mathfrak{o}_2 \leftrightsquigarrow a_4^n$
$\mathfrak{i}_3 \leftrightsquigarrow a_2$		$\mathfrak{o}_3 \leftrightsquigarrow a_1$
$\mathfrak{i}_4 \leftrightsquigarrow a_3$		$\mathfrak{o}_4 \leftrightsquigarrow a_3$
$\mathfrak{i}_5 \leftrightsquigarrow a_5$		$\mathfrak{o}_5 \leftrightsquigarrow a_5$
		$\mathfrak{o}_6 \leftrightsquigarrow a_3^h$

Now, in order to be able to train the neural network to contain new information of the form of new Horn clauses (after translating the trained neural network back to the form of a logic program) we add additional hidden layer neurons—a fixed and the same number for every neuron form the output layer that is not associated with a fact form a logic program $\mathcal{P}$. How the number of additional hidden layer neurons is established will be discussed in

the later part of the subsection. We exclude neurons associated with facts because such atoms will always be *true* in any interpretation that is a model for a logic program. In terms of neural networks this means that the output layer neurons associated with facts will always be active, because of their connection with the *truth* neuron. Afterwards, we connect additional hidden layer neurons with other neurons in the neural network in the following way:

1. Additional hidden layer neurons are connected with the output layer neurons that they were established for, i.e. if we choose to add two additional hidden layer neurons per output layer neuron, then first two additional hidden layer neurons are connected only with the first output layer neuron, the next two additional hidden layer neurons are connected only with the second output layer neuron, and so on.
2. Each input layer neuron (with the exception of the $\mathfrak{t}$ neuron) is connected with every additional hidden layer neuron, if the additional hidden layer neuron is not connected to the output layer neuron with the same *label* as the concerned input layer neuron.
3. The hidden layer neuron $\mathfrak{h}_{\mathfrak{t}}$ is connected to every output layer neuron, if it was not already connected with it or, if it is not the head of the $\mathcal{G}_{\mathcal{P}}$.

All those new additional connections have to have weight close to 0 but not 0, because of the symmetry that can cause problems during neural network training—similarly as in [14, p. 59].

The first kind of additional connections between additional hidden layer neurons and output neurons basically defines how many new Horn clauses can be established with the same atom as their head, because each additional hidden layer neuron is connected to only one output layer neuron. The second kind of additional connections serves exactly the same purpose—input neurons are connected with all additional hidden layer neurons in order to enable new Horn clauses formulation during the training process. However, there are some connections that we want to avoid, namely those which would create a connection between the same atoms and therefore, they would lead to formulation of tautological Horn clauses, that is clauses of the form:

$$a_i \leftarrow a_h, \ldots, a_i, \ldots, a_k$$

In the Figure 4.4 we can see a part of the neural network from the Figure 4.3 that is enriched by additional hidden layer neurons (two per each output layer neuron) and additional connections from input to hidden layer and from hidden to output layer. As we can see, there are no additional connections between neuron $\mathfrak{i}_1$ and additional hidden layer neurons $\mathfrak{h}_7$ and $\mathfrak{h}_8$, because both additional hidden layer neurons are connected with the output layer neuron $\mathfrak{o}_3$, which in turn is associated with the same atom as the input layer neuron $\mathfrak{i}_1$ is, what is represented by the fact that there is a recursive connection between neuron $\mathfrak{o}_3$ and $\mathfrak{i}_3$.

Finally, we add additional connections between hidden layer truth neuron $\mathfrak{h}_{\mathfrak{t}}$ and all output layer neurons that are neither labelled with the head of the

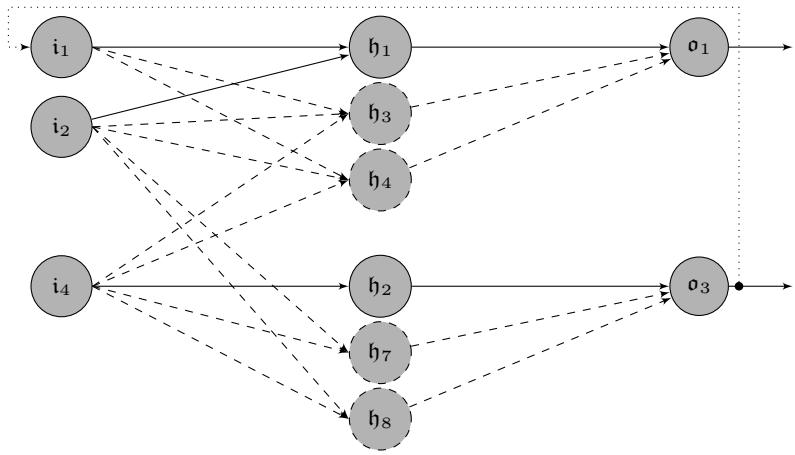

Fig. 4.4 Neural network that corresponds to the part of the logic program from the example 4.27. Black arrows denote connection with the weight W, dashed arrows denote additional connections, dotted arrows denote recursive connections with the weight 1 and dashed circles denote additional hidden layer neurons.

goal Horn clause $\mathcal{G}_{\mathcal{P}}$ nor are already connected to the $\mathfrak{h}_{\mathfrak{t}}$ neuron, i.e. they are associated with atoms that are heads of facts in logic program $\mathcal{P}$. On the one hand, we do not have to double connections for neurons that are associated with facts because they cannot become "more factive", hence the second restriction. On the other hand, we want the head of the goal Horn clause $\mathcal{G}_{\mathcal{P}}$ to be an element of the fixpoint of the immediate consequence operator $T_{\mathcal{P}}$ but we do not want just to add it to the set but instead, we want to build connections amongst neurons in the neural network that enable the head of the abductive goal $\mathcal{G}_{\mathcal{P}}$ to appear in it. Therefore, we have the first restriction.

The whole enriched neural network that results form the neural network from the Figure 4.3 can be seen in the Figure 4.5 where all of the "original" neurons and connections are depicted along with the additional hidden layer neurons (two per each output layer neuron) and additional connections.

Before we discuss the problem of the recursive connections from output layer neurons associated with atoms that have upper index h (in the Figure 4.5 the problem is marked by a black rectangle), we will show, that we do not have to modify equations that concern biases of the neurons from the previous subsection. Let us recall the Horn clause h_x form the example 4.2:

$$h_x : a_0 \leftarrow a_1, \ldots, a_k, \sim a_l, \ldots, \sim a_m$$

Enriching the neural network that is shown in the Figure 4.2 by additional hidden layer neurons and additional connections we obtain a neural network from the Figure 4.6.

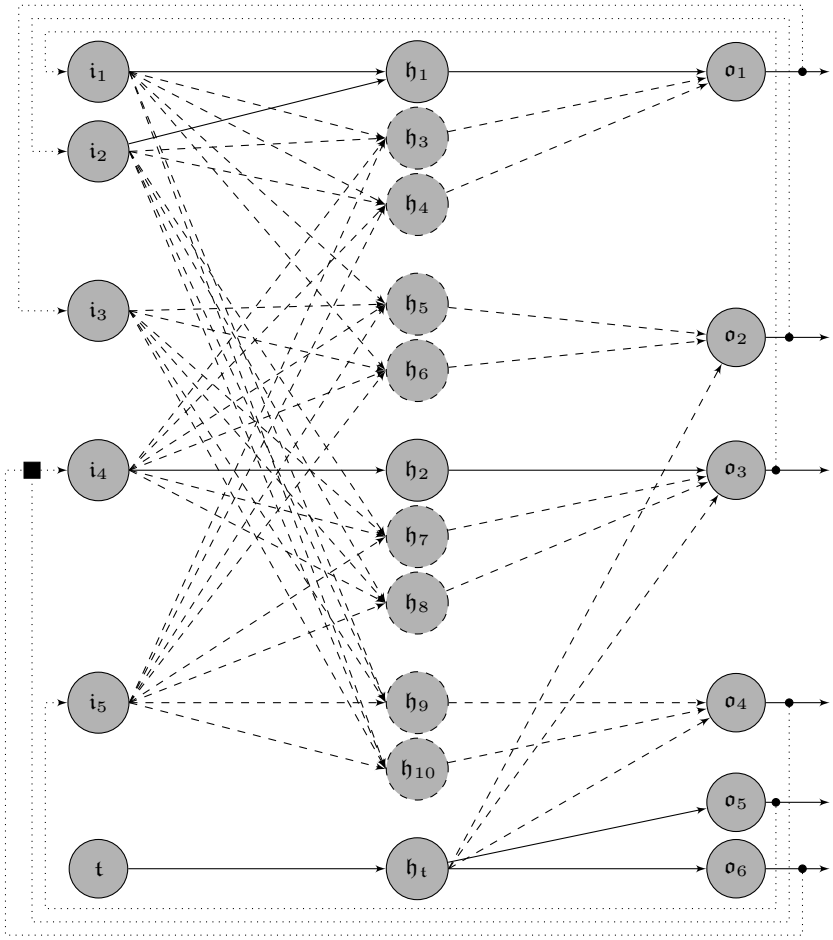

Fig. 4.5 Neural network that corresponds to the logic program from the example 4.27. Arrows denote connection with the weight W, dashed arrows denote additional connections, dotted arrows denote recursive connections with the weight 1 and dashed circles denote additional hidden layer neurons.

We do not change anything concerning the input layer neurons and what is more, we do not allow to create new connections between input layer neurons and hidden layer neurons that are not additional ones (as can be seen in the Figure 4.5 and even more clearly in the Figure 4.4). In other words, the neural network that was created at the beginning (see Figure 4.3) is not changed as far as original hidden layer neurons are concerned. Therefore, equation 4.6 that allow us to calculate the bias $b_{\mathfrak{h}_n}$ for the neuron $\mathfrak{h}_n$ can be left unchanged.

The situation seems to be different in case of the output layer neuron $\mathfrak{o}_p$. There are multiple additional connections that run from additional hidden layer neurons to the neuron $\mathfrak{o}_p$ and one additional connection from hidden

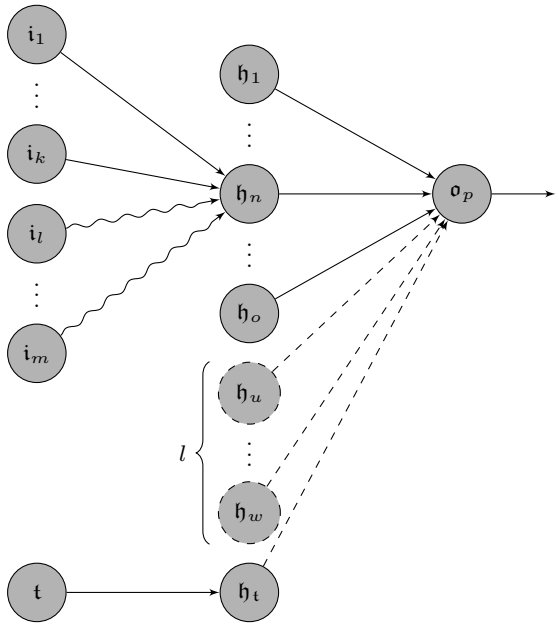

Fig. 4.6 Single Horn clause example—enriched

layer *truth* neuron $\mathfrak{h}_\mathfrak{t}$ that also serves as input for the neuron $\mathfrak{o}_p$. As in the previous subsection, we will calculate the bias for the neuron $\mathfrak{o}_p$ as average from the minimal activation potential $v^{\min}_{\mathfrak{o}_p}$, according to which the neuron $\mathfrak{o}_p$ should be considered *active*, and maximal activation potential $v^{\max}_{\mathfrak{o}_p}$, according to which the neuron $\mathfrak{o}_p$ should be considered *inactive*. Let us recall the equation 4.7:

$$b_{\mathfrak{o}_p} = \frac{v^{\min}_{\mathfrak{o}_p} + v^{\max}_{\mathfrak{o}_p}}{2}$$

We have to extend the equation 4.8 for calculation of the minimal activation potential $v^{\min}_{\mathfrak{o}_p}$ with the signal that the neuron $\mathfrak{o}_p$ receives from additional hidden layer neurons $\mathfrak{h}_u, \ldots, \mathfrak{h}_w$, and the hidden layer *truth* neuron $\mathfrak{h}_\mathfrak{t}$. We can assume that there is l additional hidden layer neurons (as it is stated in the Figure 4.6) and that each additional connection from additional hidden layer neuron with the neuron $\mathfrak{o}_p$ has some maximal weight r. In such case, the activation potential would be minimal, if all additional hidden layer neurons have their output signal -1[3]. The output signal of the hidden layer *truth* neuron $\mathfrak{h}_\mathfrak{t}$ is always equal to 1, therefore we will assume, that the weight of the additional connection that connects $\mathfrak{h}_\mathfrak{t}$ neuron with $\mathfrak{o}_p$ neuron is equal to $-r$.

[3] We could have assumed the opposite, i.e. that additional connections from additional hidden layer neurons have value $-r$ and the output signal of additional hidden layer neurons is 1.

$$\begin{aligned} v_{\mathfrak{o}_p}^{\min} &= A_{\min}W - (\mu(h_x) - 1)W + l(-1)r + (-r) = \\ &= A_{\min}W - (\mu(h_x) - 1)W - r(l+1) \end{aligned} \tag{4.28}$$

Similarly, we have to add to equation 4.9, that allows us to calculate activation potential $v_{\mathfrak{o}_p}^{\max}$, information about additional connections from additional hidden layer neurons and the *truth* neuron. In this case we assume that all of the l additional hidden layer neurons are in 1 state and that they are connected with the neuron $\mathfrak{o}_p$ with weight r. Additionally, the *truth* neuron is also connected with the neuron $\mathfrak{o}_p$ with the weight r.

$$\begin{aligned} v_{\mathfrak{o}_p}^{\max} &= -A_{\min}\mu(h_x)W + lr + r = \\ &= -A_{\min}\mu(h_x)W + r(l+1) \end{aligned} \tag{4.29}$$

After substitution for activation potentials $v_{\mathfrak{o}_p}^{\min}$ and $v_{\mathfrak{o}_p}^{\max}$ to the equation 4.7 we obtain the same result as the equation 4.10 in the previous subsection, i.e.:

$$b_{\mathfrak{o}_p} = \frac{(1 + A_{\min})(1 - \mu(h_x))}{2} W$$

For the same reason the equation 4.13 that defines the boundary condition for $A_{\min}$ value is can be used here:

$$A_{\min} > \frac{\max(K_{\mathcal{P}} \cup Mu_{\mathcal{P}}) - 1}{\max(K_{\mathcal{P}} \cup Mu_{\mathcal{P}}) + 1}$$

Now we are going to calculate the condition for the weight of the connections W. Since there is no change for the hidden layer neurons the condition in equation 4.17 remains the same:

$$W \geq -\frac{2}{\beta} \cdot \frac{\ln(1 - A_{\min}) - \ln(1 + A_{\min})}{\mathsf{k}(h_x)(A_{\min} - 1) + A_{\min} + 1}$$

In the light of the assumption that every weight of a connection in the neural network is equal to W or $-W$ we can generalize the above given equation for the whole logic program:

$$W \geq -\frac{2}{\beta} \cdot \frac{\ln(1 - A_{\min}) - \ln(1 + A_{\min})}{\max(K_{\mathcal{P}})(A_{\min} - 1) + A_{\min} + 1} \tag{4.30}$$

This is not the case, if we take under consideration output layer neurons. On the one hand, the activation of the neuron $\mathfrak{o}_p$ should be greater than the $A_{\min}$ value, if we consider neuron $\mathfrak{o}_p$ to be *true*. Hence, the first condition is the same as it is stated in the equation 4.19:

$$v_{\mathfrak{o}_p} \geq -\frac{1}{\beta} \ln\left(\frac{1 - A_{\min}}{1 + A_{\min}}\right)$$

On the other hand, if the neuron $\mathfrak{o}_p$ is considered to be *true*, then the activation potential $v_{\mathfrak{o}_p}^{\min}$ should be grater than the value obtained when the $\mathfrak{h}_n$ neuron is in minimal activation state, i.e. its output signal is equal to $A_{\min}$, the rest of the hidden layer neurons that are not additional ones and are connected with the neuron $\mathfrak{o}_p$ have their output signal equal to -1, all of the additional hidden layer neurons that are connected with the $\mathfrak{o}_p$ neuron have output signal equal to -1 and the weight of their additional connections is equal to r, the weight of the additional connection between hidden layer *truth* neuron $\mathfrak{h}_\mathfrak{t}$ and the neuron $\mathfrak{o}_p$ is equal to $-r$, and finally, minus the bias $b_{\mathfrak{o}_p}$:

$$v_{\mathfrak{o}_p} \geq A_{\min}W + (\mu(h_x) - 1)(-1)W + l(-1)r + (-r) - b_{\mathfrak{o}_p}$$

and after substitution for the bias $b_{\mathfrak{o}_p}$ we obtain:

$$v_{\mathfrak{o}_p} \geq A_{\min}W - (\mu(h_x) - 1)W - r(l+1) - \frac{(1 + A_{\min})(1 - \mu(h_x))}{2}W \quad (4.31)$$

Combining both inequalities 4.31 and 4.19 gives us the following inequality:

$$A_{\min}W - (\mu(h_x) - 1)W - r(l+1) - \frac{(1 + A_{\min})(1 - \mu(h_x))}{2}W \geq -\frac{1}{\beta}\ln\left(\frac{1 - A_{\min}}{1 + A_{\min}}\right) \quad (4.32)$$

from which we obtain the condition for the weight W, given that $W > 0$:

$$W \geq -\frac{2}{\beta} \cdot \frac{\ln(1 - A_{\min}) - \ln(1 + A_{\min}) + r(l+1)}{\mu(h_x)(A_{\min} - 1) + A_{\min} + 1} \quad (4.33)$$

We can generalise the condition for the whole logic program as in the case of the equation 4.30:

$$W \geq -\frac{2}{\beta} \cdot \frac{\ln(1 - A_{\min}) - \ln(1 + A_{\min}) + r(l+1)}{\max(Mu_{\mathcal{P}})(A_{\min} - 1) + A_{\min} + 1} \quad (4.34)$$

In case of the logic program $\mathcal{P}_p^{\mathsf{dn}(h_g)}$ from the example 4.27 and the neural network structure that was created on its basis, now is the time to calculate the weight W of connections and the $A_{\min}$ value along with biases for hidden and output layer neurons. Given the Horn clauses from logic program $\mathcal{P}_p^{\mathsf{dn}(h_g)}$:

$h_1 : a_2 \leftarrow a_1, a_4^n$	$\mathsf{k}(h_1) = 2$	$\mu(h_1) = 1$
$h_2 : a_1 \leftarrow a_3$	$\mathsf{k}(h_2) = 1$	$\mu(h_2) = 1$
$h_3 : a_5 \leftarrow$	$\mathsf{k}(h_3) = 0$	$\mu(h_3) = 1$
$h_4 : a_3^h \leftarrow$	$\mathsf{k}(h_4) = 0$	$\mu(h_4) = 1$

we obtain the following sets:

$$\begin{aligned} K_{\mathcal{P}_p^{\mathsf{dn}(h_g)}} &= \{2,1,0\} \\ Mu_{\mathcal{P}_p^{\mathsf{dn}(h_g)}} &= \{1\} \end{aligned} \tag{4.35}$$

and using equation 4.13 we obtain:

$$A_{\min} > \frac{1}{3} \tag{4.36}$$

Let us assume that $A_{\min} = 0.5$. Let us further assume that $\beta = 1$, therefore from equation 4.30 we obtain the following inequality:

$$W \geq 4.396 \tag{4.37}$$

At the beginning of this subsection we decided that there are two additional neurons per output layer neuron ($l = 2$). In order to be able to calculate the second inequality for weight W from equation 4.34 we have to assume some value for r, e.g. $r = 0.05$. Therefore, we have the following:

$$W \geq 1.898 \tag{4.38}$$

It turns out that for this particular example we can use the same weight $W = 4.5$ as in the example form the previous subsection that concerned logic program 4.1 and neural network 4.1 obtained from it. Using $A_{\min}$ and W values we can calculate biases for hidden and output layer neurons. As a result we obtain almost all information that we need to complete the neural network, which is shown below:

$$\begin{array}{lll}
\mathfrak{i}_1 = \langle g(\cdot), 0, a_1, 0.5\rangle & \mathfrak{h}_1 = \langle h(\cdot), 3.375, h_1, 0.5\rangle & \mathfrak{o}_1 = \langle h(\cdot), 0, a_2, 0.5\rangle \\
\mathfrak{i}_2 = \langle g(\cdot), 0, a_4^n, 0.5\rangle & \mathfrak{h}_2 = \langle h(\cdot), 0, h_2, 0.5\rangle & \mathfrak{o}_2 = \langle h(\cdot), 0, a_4^n, 0.5\rangle \\
\mathfrak{i}_3 = \langle g(\cdot), 0, a_2, 0.5\rangle & \mathfrak{h}_{\mathfrak{t}} = \langle 1, 0, ht, 0.5\rangle & \mathfrak{o}_3 = \langle h(\cdot), 0, a_1, 0.5\rangle \\
\mathfrak{i}_4 = \langle g(\cdot), 0, a_3, 0.5\rangle & & \mathfrak{o}_4 = \langle h(\cdot), 0, a_3, 0.5\rangle \\
\mathfrak{i}_5 = \langle g(\cdot), 0, a_5, 0.5\rangle & & \mathfrak{o}_5 = \langle h(\cdot), 0, a_5, 0.5\rangle \\
\mathfrak{t} = \langle 1, 0, t, 0.5\rangle & & \mathfrak{o}_6 = \langle h(\cdot), 0, a_3^h, 0.5\rangle
\end{array}$$

The additional hidden layer neurons are not listed above, because they are all the same except for the *label*:

$$\mathfrak{h}_i = \langle h(\cdot), b_a, ha_n, 0\rangle$$

where $i \in [3, 10]$ and $n \in [1, 8]$ (there are 2 additional hidden layer neurons per output layer neuron and we have 4 output layer neurons that are not associated with facts, therefore there are 8 additional hidden layer neurons).

There is one last part of the neural network that needs to be explained. We have to deal with the problem of recursive connections and absence of some of the input layer neurons that would be associated with atoms with upper

index h. Both problems are strictly connected with the definition 2.46 of the immediate consequence operator $T_{\mathcal{P}}$ and the way it maps interpretations into interpretations for a given logic program. Let us recall that the head of a given Horn clause h_i belongs to the set that is the result of applying immediate consequence operator $T_{\mathcal{P}}$ to an interpretation $I_{\mathcal{P}}$, if all atoms from the positive part of the body of clause h_i, or their counterparts with the upper index h, belong to the interpretation $I_{\mathcal{P}}$ and at the same time, no atom that belongs to the negative part of the body of clause h_i is an element of the interpretation $I_{\mathcal{P}}$. Now, the problem concerns the "or" part, namely, when it is the case that there are two output layer neurons where one is associated with atom a_j and the other one is associated with its counterpart a_j^h. We could just place in the input layer a neuron that is associated with the atom a_j^h. However, then we would have to face the next problem: the immediate consequence operator $T_{\mathcal{P}}$ cannot differentiate between an atom a_j and its counterpart a_j^h as far as the body of a Horn clause is concerned, therefore the neural network that is supposed to model the way the $T_{\mathcal{P}}$ operator works should also "not differentiate" between both input neurons. Having neurons in the input layer for both: atom a_j and its counterpart a_j^h we would have to modify the structure of the network in such a way that it would guarantee that when all other conditions are met, the output layer neuron associated with the head of a Horn clause containing atom a_j is considered active if either the neuron associated with atom a_j or the neuron associated with atom a_j^h is considered active. On the one hand, such modifications would lead to other complications concerning training of the neural network or interpretation of the translated neural network into a logic program. On the other hand, when we consider the way the immediate consequence operator $T_{\mathcal{P}}$ works for a logic program $\mathcal{P}$ we treat an exemplary atom a_j from the interpretation $I_{\mathcal{P}}$ as the same atom a_j, if it belongs to the set that is a result of the application of $T_{\mathcal{P}}$ operator to the interpretation $I_{\mathcal{P}}$. However, the situation is different after the translation of the logic program $\mathcal{P}$ into a neural network $\mathfrak{N}$. An input layer neuron associated with atom a_j from the interpretation $I_{\mathcal{P}}$ is independent from a neuron from the output layer that is associated with the same atom a_j and here interpreted as an element of the set resulting from application of $T_{\mathcal{P}}$ operator to the interpretation $I_{\mathcal{P}}$. As we will see in the Section 4.2 about neural network training the recursive connections serve only to carry the information about the state of the neuron from the output layer, that is associated with a given atom, to the neuron from the input layer, that is associated with the same atom, in order to set the state of the input layer neuron to be the same as the state of the output layer neuron. This is the reason for which recurrent connections are left untouched while neural network training. Having this perspective in mind it seems that the solution to the problem of atoms with the upper index h should be organised in such a way that it would not influence the part of the neural network that will be trained. For that reason I decided to add additional neurons between the

output and the input layer, which will mediate the output signal from the output layer neurons on the recurrent connections to the input layer neurons.

Let us examine a part of the neural network from example 4.27 where the problem occurs, i.e. between atoms a_3 (associated with neuron $\mathfrak{i}_4$ and $\mathfrak{o}_4$) and a_3^h (associated with neuron $\mathfrak{o}_5$). Figure 4.7 shows part of the neural network from the example 4.5 where the problematic point is marked with a black rectangle.

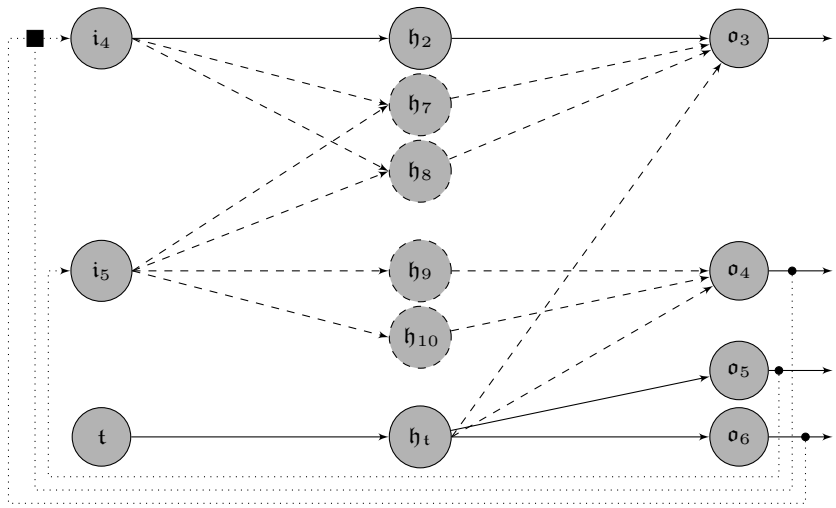

Fig. 4.7 Part of the neural network from the example 4.5 with the problematic point denoted by black rectangle.

Our goal is to merge signals from two recurrent connections running from neurons $\mathfrak{o}_4$ and $\mathfrak{o}_5$ into one signal that serves as an input for neuron $\mathfrak{i}_4$. There are two conditions that have to be fulfilled:

1. The output signal of neuron $\mathfrak{i}_4$ has to be between $[-1, 1]$.
2. Neuron $\mathfrak{i}_4$ should be always active.

The first condition is concerned with the fact that the activation function in the input layer neurons is the identity function $g(x)$ and when the input signal for such neuron exceeds 1 it can "dominate" the rest of the input neurons connected to the same hidden layer neuron, since the calculations of the $A_{\min}$ and W values, and what follows, biases for hidden and output layer neurons assumed that the maximal output value of any input layer neuron is equal to 1. Therefore, it could be the case that a hidden layer neuron would be considered active, if not all of the input layer neurons that are connected to it with a positive weight W are considered to be active, or even if one of the input layer neurons that is connected to it with a negative weight W is considered to be active. In the first case the situation for the logic program

would look as if the head of a Horn clause could belong to the set resulting from applying the immediate consequence operator to an interpretation that does not contain all of the positive atoms from the body of that Horn clause. In the second case the result would be similar, but now the interpretation that serves as an input for the immediate consequence operator contain one of the atoms that occur negatively in the body of a concerned Horn clause.

In order to be sure that the output signal of the input neurons has its values between -1 and 1, we can act in two different ways:

1. We can add an additional neuron $\mathfrak{r}_a$ that has neurons $\mathfrak{o}_4$ and $\mathfrak{o}_6$ as inputs, and neuron $\mathfrak{i}_4$ as a receiver of the output signal, that would serve as a summing junction for output signals from those two output layer neurons.
2. We can can change the activation function in the input layer neurons from identity function $g(x)$ to a function which returns values from the interval $[-1, 1]$.

We have already stated before that we want to minimise changes in the part of the neural network that will be influenced by the training process. Therefore we chose the first solution, i.e. we add an additional neuron $\mathfrak{r}_a$ in place of the black rectangle from the Figure 4.7 that will combine signals from output layer neurons that are associated with atoms and counterparts of those atoms. We propose the following activation for neuron $\mathfrak{r}_a$:

$$y(x) = \begin{cases} -1 & \text{if } x \leq -1 \\ x & \text{if } x \in (-1; 1) \\ 1 & \text{if } x \geq 1 \end{cases} \tag{4.39}$$

As it turns out, this solution allows us to manage also the second condition in an easy and clean manner. Let us recall that we want the "permanent activation" of the neuron from the input layer. This requirement is concerned with the fact that if this problem occurs for neurons from input and output layer that are associated with an atom a_i it is because of the existence of the output layer neuron that is associated with the counterpart a_i^h of the atom a_i. We know that counterparts of atoms are added to the logic program only as facts. Therefore, it is always the case that the neuron that is associated with the problematic atom a_i^h is located only in the output layer and additionally, it will be always connected with the *truth* neuron. This is consistent with the way the immediate consequence operator $T_{\mathcal{P}}$ works for a logic program $\mathcal{P}$ that is enriched by facts in the form of counterparts of atoms that are already in the program $\mathcal{P}$. Starting from arbitrary interpretation the $T_{\mathcal{P}}$ operator will return the set of atoms that contains all atoms with the upper index h. What follows, in consecutive iterations of $T_{\mathcal{P}}$ operator, if all other conditions are met, we obtain the head of a Horn clause that contains atom a_i in the body even if atom a_i was not an element of the interpretation that was an argument for the $T_{\mathcal{P}}$ operator, but instead the fact a_i^h was present.

Neuron associated with a given atom is considered to be *true* when its activation is greater than the $A_{\min}$ value. The additional neuron $\mathfrak{n}_a$ that combines signals from output layer neurons associated with atom a_i and its counterpart a_i^h is connected with those neurons, as well as with the input layer neuron associated with atom a_i, and the weight of those connections is equal to 1. Therefore, the activation of neuron $\mathfrak{n}_a$ will be the same as the activation of the input layer neuron associated with atom a_i, if we do not want to add some bias to the input layer neuron. The lowest possible activation of the additional $\mathfrak{n}_a$ neuron would be equal to $A_{\min} - 1$[4], because the lowest activation of the output layer neuron associated with atom a_i is equal to -1 and the lowest activation of the output layer neuron associated with atom a_i^h is equal to $A_{\min}$.

The easy solution that will keep the activation of the input layer neuron above the $A_{\min}$ value concerns slight modification of the $y(x)$ function described in 4.39:

$$k(x) = \begin{cases} A_{\min} + A^a_{\min} & \text{if } x \leq A_{\min} + A^a_{\min} \\ x & \text{if } x \in (A_{\min} + A^a_{\min}; 1) \\ 1 & \text{if } x \geq 1 \end{cases} \tag{4.40}$$

The $A^a_{\min}$ factor is needed because, as we have stated above, a neuron is considered to be *true* when its activation is greater than the $A_{\min}$ value.

For every pair of an atom and its counterpart with the upper index h the situation looks the same. Therefore, instead of black rectangles we place additional neuron between the output and input layer with $k(\cdot)$ as activation function, bias equal to 0, labelled oi_n (where $n \in \mathbb{N}$) and 0 instead of $A_{\min}$ value, since there is no sense to talk about this neuron in *truth/false* categories:

$$\mathfrak{r}_n = \langle k(\cdot), 0, oi_n, 0 \rangle \tag{4.41}$$

Figure 4.8 depicts how the connections run for the neuron $\mathfrak{r}_n$.

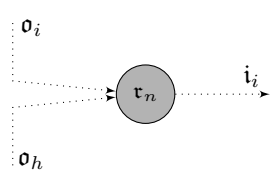

Fig. 4.8 Neuron that combines signals from output layer neurons in the problematic place marked in the example 4.5 by a black rectangle. All connections are dotted, i.e. treated as recursive, which means they have weight equal to 1. Labels of the neurons that are connected with the $\mathfrak{r}_n$ neuron are indicated near each connection.

[4] In fact, the activation of the additional neuron will be slightly greater than the value $A_{\min} - 1$, because the output layer neuron associated with atom a_i^h is always considered *true*, therefore its activation is greater than $A_{\min}$ value.

It is worth noting that such solution allows us to maintain at least part of the information about the output layer neuron associated with an atom a_i for the input layer neuron that is also associated with the atom a_i. The input layer neuron reflects the activation of the neuron $\mathfrak{r}_n$, because it has identity function as activation function. As we can see, the function $k(\cdot)$ from equation 4.40 keeps the input layer neuron in a state in which in is considered to be *true*. But the state of our input layer neuron (and every other) is more complex than the logical division *true*/*false*, i.e. although given neuron is for example considered to be *true* it can be activated in range $[A_{\min} + A^a_{\min}; 1]$. Therefore, different states of the output layer neuron associated with the atom a_i would reflect in a different states of the input layer neuron that is associated with the same atom preserving that it would be considered *true*.

At this point we have a complete transformation of a logic program and an abductive goal into a neural network. It appears that there are multiple factors and coefficients that influence the shape of the neural network as well as some of the properties of parts of the neural network. Those factors and coefficients can influence the process of neural network training therefore we want to take them into account while translating a logic program and an abductive goal into a neural network. We can list all of the factors that we want to consider:

- β — coefficient in the $h(\cdot)$ activation function described by equation 3.12,
- l — number of additional neurons in the hidden layer per output layer neuron,
- r — limits for the range for the weight of additional connections r (additional connections have random values in the range $[-r, r]$ without 0),
- b_a — bias for additional hidden layer neurons,
- $A^a_{\min}$ — coefficient for the $k(\cdot)$ activation function,
- W^f — weight W factor,
- $A^f_{\min}$ — $A_{\min}$ value factor.

The first factor, i.e. β is concerned with the construction of the neurons that establish hidden and output layer of the neural network. The next three factors, i.e. l, r and b_a are concerned with the addition of the neurons in the hidden layer and additional connections between neurons from input and hidden layer, and hidden and output layer. Factor $A^a_{\min}$ is added because of the aroused problem with the neurons associated with atoms and their counterparts with upper index h. Finally, two last factors, i.e. W^f and $A^f_{\min}$ are added, because equations 4.13, 4.30 and 4.34 allow only to calculate the limit for $A_{\min}$ and W values. In order to be able to control their values we will calculate them as:

$$A_{\min} = A^b_{\min} + A^f_{\min} \tag{4.42}$$

$$W = W^b + W^f \tag{4.43}$$

where W^b and $A^b_{\min}$ will be calculated by means of mentioned equations.

Now we can finally describe the whole translation algorithm. We assume, that the starting point is a logic program $\mathcal{P}$ and an abductive goal $\mathcal{G}_{\mathcal{P}}$ for that logic program, and that we have provided all factors and coefficients mentioned above:

1. Create positive version of the logic program $\mathcal{P}$, denoted by $\mathcal{P}_p$, and positive version of the abductive goal $\mathcal{G}_{\mathcal{P}}$, denoted by $\mathsf{dn}(\mathcal{G}_{\mathcal{P}})$.
2. Create modified version of the logic program $\mathcal{P}_p$ w.r.t. the abductive goal $\mathsf{dn}(\mathcal{G}_{\mathcal{P}})$, i.e. $\mathcal{P}_p^{\mathsf{dn}(\mathcal{G}_{\mathcal{P}})}$.
3. Calculate $A^b_{\min}$ (by means of equation 4.13) and W^b (by means of equations 4.22 and 4.30—choose greater value) values and what follows, $A_{\min}$ and W values (by means of equations 4.42 and 4.43, respectively).
4. Add the *truth* neuron $\mathfrak{t}$ to the input layer and neuron $\mathfrak{h}_{\mathfrak{t}}$ to the hidden layer.
5. For every Horn clause from logic program $\mathcal{P}_p^{\mathsf{dn}(\mathcal{G}_{\mathcal{P}})}$:
   a. Add output layer neuron associated with the head of the Horn clause; exceptions:
      - there already exists an output layer neuron that is associated with that atom.
   b. If the Horn clause is a fact, then connect added output layer neuron with the $\mathfrak{h}_{\mathfrak{t}}$ neuron. Otherwise, add a hidden layer neuron and connect it with the added output layer neuron.
   c. Add input layer neuron for each atom from the body of the Horn clause and for the head of the Horn clause; exceptions:
      - there already exists an output layer neuron that is associated with that atom,
      - the head of the Horn clause has upper index h and the counterpart of this atom is an element of the Herbrand base $B_{\mathcal{P}_p^{\mathsf{dn}(\mathcal{G}_{\mathcal{P}})}}$.
   d. Connect all neurons from the input layer that are associated with atoms from the body of the Horn clause with the hidden layer neuron (if atom is preceded with negation, then the weight of the connection is $-W$).
   e. Add output layer neuron for each atom from the body of the Horn clause; exceptions:
      - there already exists an output layer neuron that is associated with that atom.
6. For each output layer neuron add l additional hidden layer neurons with bias b_a and $A_{\min}$ value; exceptions:
   - output layer neuron represents a fact.
7. Add additional connections between additional hidden layer neurons and proper output layer neurons. Weight of the additional connections is a random value from range $[-r, r]$ without 0.
8. Add additional connections between $\mathfrak{h}_{\mathfrak{t}}$ neuron and output layer neurons; exceptions:

- output layer neuron represents a fact,
- output layer neuron is associated with the head of abductive goal $\mathcal{G}_{\mathcal{P}}$.

9. Add additional connections between the input layer neurons and the hidden layer neurons; exceptions:
   - the input layer neuron has the same *label* as the output layer neuron that is connected with the hidden layer neuron,
   - hidden layer neuron is $\mathfrak{h}_{\mathfrak{t}}$ neuron.
10. Add additional neurons between the output and the input layer for all those input layer neurons that are associated with atoms for which there exist counterparts with upper index h in the Herbrand base $B_{\mathcal{P}_{\mathcal{P}}^{\mathtt{dn}(\mathcal{G}_{\mathcal{P}})}}$.
11. Add recursive connections, i.e. connections with weight equal to 1, for the following neurons:
    - between the output layer neurons associated with atoms for and those associated with counterparts for those atoms, and additional neurons added in the previous step,
    - between additional neurons added in the previous step and proper input layer neurons,
    - all other output layer neurons and input layer neurons that have the same *label*.

    The detailed translation algorithm is given in the Appendix 8.

4.1.2 Translation from neural networks to logic programs

Andrews et al. [2, p. 376] proposed the method of classification for different strategies of rules (of any kind) extraction from trained neural networks, which consists of five dimensions:

1. The expressive power of the extracted rules.
2. The *translucency* of the view taken within the rule extraction technique of the underlying artificial neural network units.
3. The extent to which the underlying artificial neural network incorporates specialised training regimes.
4. The *quality* of the extracted rules.
5. The algorithmic *complexity* of the rule extraction/rule refinement technique.

The first dimension concerns the form of the extracted rules. As the authors underline, in majority of cases rules are of the form *if*... *then*... *else* and the difference between strategies lies only in the field of values used in the logic, i.e. standard two-valued Boolean logic or fuzzy logic.

According to the second dimension, there are three ways of extracting rules from a given neural network: *pedagogical*, *decompositional* and *eclectic*. The difference between these approaches is in the treatment of the neural network. The first method maps all possible input states into output states, regardless of the neural network hidden structures. Thus, the neural network is treated as a *black box* equipped with some function that allows to generate an output state from any given input state, and our goal is to model this function by the mentioned mapping. Andrews et al. [2] describe the neural network in this approach as *opaque*, since we do not want to have access to the internal structure of the network.

The second approach, i.e. the decompositional strategy, uses internal structure of the neural network to generate the set of rules that reflects the way the neural network works. Algorithms operating in such approach create rules for every hidden and output layer neuron alone—not for the whole output layer in the neural network, like it was described in the previous approach. Therefore, in terms of translucency, a neural network is *transparent* in this setting. The eclectic group contains approaches that use techniques from both, the pedagogical and the decompositional strategies.

It could be the case that the rule extraction technique requires special "training regime" for artificial neural network. Therefore, the third dimension concerns the *portability* of the rule extraction techniques across various artificial neural networks architectures [2, p. 377].

In the fourth dimension the translation algorithm is evaluated according to the quality of the translations, where the quality of a translation is established by the following criteria: accuracy (ability of correct classification of the previously unseen examples), fidelity (coverage of the whole information embodied in the neural network), consistency (sets of rules extracted from the neural networks under different training sessions produce the same classifications of unseen examples) and comprehensibility (the size of the set of rules and the size of the rules). What should be noted here is the comment from Andrews et al. "on the issue of rule quality in general and rule comprehensibility in particular [...] that the focus of discussion is exclusively on rule syntax and not on the more problematic area of rule semantics" [2, p. 377].

Finally, the translation algorithm should be evaluated also for its complexity. In other words, the more efficient the algorithm is, the better evaluation it gets.

We are going to use the approach described by Garcez et al. [14, ch. 5] at the beginning of the chapter, which is placed in the pedagogical strategies of translating neural networks into sets of rules. According to the five dimensions given by Andrews et al. [2]:

1. The set of rules obtained after translation consists of propositional Horn clauses and a two-valued logic.
2. The neural network is treated as a black box and our task is to map every possible combination of the activation of input layer neurons into the activation of the output layer neurons.

3. The algorithm will be used only with specific three-layered neural networks, that were described earlier.
4. The algorithm is sound and complete.
5. The complexity of the algorithm is 2^n, where n is the number of the input layer neurons.

The choice of the logic and form of the Horn clauses for the extracted rules is dictated by our goal, i.e. we want to compare the obtained results with the initial knowledge base, which in turn is formalised in the language of propositional normal logic programs.

Garcez et al. [14, ch. 5] chose to implement pedagogical approach, but in order to reduce the search space, which for every combination of the input layer neurons states is equal to 2^n, where n is the number of the input layer neurons, they propose a number of improvements that allow to narrow it down. This is possible due to the nature of the translated neural networks, i.e. the input layer neurons will be checked only for two states: 1 and -1 (which correspond to the *true* and *false* states, respectively), and neurons from the hidden and output layer have bipolar sigmoid activation function, and additionally, they are "equipped" with a special value $A_{\min}$ that allow to decide if an atom associated with a given neuron should be mapped to *true* or *false*. This is also the reason why the translation algorithm is designed only for a certain kind of neural networks and why some information saved in the neural network is "lost" due to the translation. It should be noted that although improvements proposed by Garcez et al. decrease the search space for the majority of problems, in the worst case the complexity is still 2^n.

Similarly as Garcez et al. [14, ch. 5], we are going to use the pedagogical approach, because "only pedagogical approaches can guarantee that the knowledge extracted is equivalent to the network, i.e. that the extraction process is sound and complete" [14, p. 116]. The difference is that after the extraction of the rules from a given neural network we are going to simplify the logic program.

Having established all properties of the translation algorithm the problem of finding a set of Horn clauses that can be associated with a given neural network can be stated in a general form as follows:

> Given a particular set of weights W_{ij} and thresholds θ_i, resulting from a training process on a neural network, find for each input vector $\bar{\imath}$, all the outputs o_j in the corresponding output vector $\bar{o}$ such that the activation of o_j is greater than $A_{\min}$, where $A_{\min} \in (0, 1)$ is a predefined value. [14, p. 114]

Some of the improvements to the translation algorithm proposed by Garcez et al. that reduce the search space can be described on the level of the language of the propositional logic programs. The first one, which is stated in the Proposition 4.1, concerns the following intuition: given two Horn clauses h_i and h_j with the same head ($head(h_i) = head(h_j)$) and the body of one Horn clause as a subset of the body of the other ($body(h_i) \subseteq body(h_j)$), we can create two logic programs—one with both Horn clauses ($\mathcal{P}_k = \{h_i, h_j\}$)

and one with the Horn clause with the "smaller" body ($\mathcal{P}_l = \{h_i\}$)—which cannot be distinguished as long as interpretations are concerned. In other words, all and only those interpretations that are models for the first logic program are all and only models for the second logic program.

Proposition 4.1. *Let us assume that* $h_i : a_k \leftarrow l_l, \ldots, l_m$ *and* $h_j : a_k \leftarrow l_l, \ldots, l_m, \ldots, l_n$ *are Horn clauses, where* $body(h_i) \subseteq body(h_j)$*. Let us further assume that* $\mathcal{P}_1 = \{h_i, h_j\}$ *and* $\mathcal{P}_2 = \{h_i\}$ *are normal logic programs. An interpretation* I *is a model for* $\mathcal{P}_1$ *iff* I *is a model for* $\mathcal{P}_2$*.*

Proof. ($\rightarrow$) If interpretation I is a model for $\mathcal{P}_1$, i.e. a set of Horn clauses, then by by Definition 2.23 every Horn clause from $\mathcal{P}_1$ is mapped to *true*, i.e. I is a model for h_i and h_j—clauses from $\mathcal{P}_1$. That means that I is a model for every Horn clause from $\mathcal{P}_2$, since h_i is the only Horn clause that is an element of $\mathcal{P}_2$.

($\leftarrow$) If interpretation I is a model for $\mathcal{P}_2$, then by Definition 2.23 h_i, which is the only element of $\mathcal{P}_2$, is mapped to *true*. In such situation, by Corollary 2.10, we have the following three possibilities:

1. $a_k \in I$, therefore by Corollary 2.10 h_i and h_j are mapped to *true*, and by Definition 2.23 interpretation I is a model for $\mathcal{P}_1$.
2. There is a positive literal l_o in the $body(h_i)$ and $l_o \notin I$. Since $body(h_i) \subseteq body(h_j)$, then $l_o \in body(h_j)$ and what follows, by Corollary 2.10 h_i and h_j are mapped to *true*, and by Definition 2.23 interpretation I is a model for $\mathcal{P}_1$.
3. There is a negative literal l_o in the $body(h_i)$ and $\mathsf{b}(l_o) \in I$. Since $body(h_i) \subseteq body(h_j)$, then $l_o \in body(h_j)$ and what follows, by Corollary 2.10 h_i and h_j are mapped to *true*, and by Definition 2.23 interpretation I is a model for $\mathcal{P}_1$. $\square$

The Proposition 4.1 serves as a base for the definition of the first simplification mechanism for logic programs, i.e. if a logic program $\mathcal{P}$ contains two Horn clauses: h_j and h_k, where the body of the clause h_j is a subset of the body of the clause h_k, and both Horn clauses have the same head, then we remove the clause h_k from the logic program $\mathcal{P}$.

Definition 4.7. Let $\mathcal{P}_i$ be a normal logic program and $\mathbb{P}$ be a family of logic programs. The mapping $\mathsf{simp1} : \mathbb{P} \longrightarrow \mathbb{P}$ is defined as follows:

$$\mathsf{simp1}(\mathcal{P}_i) = \begin{cases} \mathcal{P}_i \setminus \{h_k\} & \text{if there is } h_j \in \mathcal{P}_i \text{ such that:} \\ & body(h_j) \subseteq body(h_k) \\ \mathcal{P}_i & \text{otherwise} \end{cases}$$

It appears that there is another reason for the reduction of the number of Horn clauses in a logic program $\mathcal{P}$, namely the situation when program $\mathcal{P}$ contains two Horn clauses: h_j and h_k, where both clauses have the same head, while the body of the clause h_j differs from the body of the clause

h_k only in such a way, that the symmetric difference between them is a set of complementary literals. Let us assume for example that the body of the clause h_j contains an atom a_x and a literal $\sim a_y$, if the body of the clause h_k contains a literal $\sim a_x$ and an atom a_y instead of the atom a_x and the literal $\sim a_y$, and the rest of literals the same as the body of Horn clause h_j, then the symmetric difference between bodies of those two Horn clauses yields the following set of complementary literals: $\{a_x, \sim a_x, a_y, \sim a_y\}$. In such a situation we can remove both Horn clauses from the program $\mathcal{P}$ and instead add a "new" one that has the same head as both former clauses h_j and h_k, and the body consisting only of those literals that were common elements between both clauses. In other words, the "new" Horn clause has the body of the clause h_j with removed literal $\sim a_x$ and atom a_y, which is the same as the body of the clause h_k with removed atom a_x and literal $\sim a_y$. Let us now define the mapping that returns the symmetric difference between the bodies of two Horn clauses.

Definition 4.8. Let h_i and h_j be normal Horn clauses, *Hcl* be a family of normal Horn clauses and $\mathbb{L}$ be a family of sets of literals. The mapping $\mathsf{dif} : Hcl^2 \longrightarrow \mathbb{L}$ is defined as follows:

$$\mathsf{dif}(h_i, h_j) =_{df} \{l_x \mid l_x \in body(h_i) \cup body(h_j),\ l_x \notin body(h_i) \text{ or } l_x \notin body(h_j)\}$$

We can use the newly defined mapping to establish a property of two Horn clauses that have the same head and their bodies differ in such a way that the symmetric difference between them returns the set of complementary literals. We will call such clauses *complementary*.

Definition 4.9. Let h_i and h_j be a normal Horn clauses, and a_x an atom. We call h_i and h_j *complementary* iff $head(h_i) = head(h_j)$, $\mathsf{dif}(h_i, h_j) \neq \emptyset$ and one of the following conditions is satisfied:

- if $a_x \in \mathsf{dif}(h_i, h_j)$ and $a_x \in body(h_i)$, then $\sim a_x \in \mathsf{dif}(h_i, h_j)$ and $\sim a_x \in body(h_j)$,
- if $\sim a_x \in \mathsf{dif}(h_i, h_j)$ and $\sim a_x \in body(h_i)$, then $a_x \in \mathsf{dif}(h_i, h_j)$ and $a_x \in body(h_j)$.

For example, the following two Horn clauses are complementary:

$$\begin{aligned} h_1 &: a_1 \leftarrow a_2, \sim a_3, a_4, \sim a_5 \\ h_2 &: a_1 \leftarrow a_2, \sim a_3, \sim a_4, a_5 \end{aligned} \tag{4.44}$$

Now we can prove that as far as interpretations are concerned, there is no difference between a logic program $\mathcal{P}_m$ that contains two complementary Horn clauses h_i and h_j, and a logic program $\mathcal{P}_n$ that contains a Horn clause h_k, which has the same head as the complementary clauses h_i and h_j, and the body of the clause h_k is equal to the body of either, clause h_i or h_j, minimised by the symmetric difference between both complementary clauses.

Proposition 4.2. *Let h_i and h_j be normal Horn clauses that are complementary and h_k is also a normal Horn clause, where $head(h_k) = head(h_i) = head(h_j)$ and $body(h_k) = body(h_i) \setminus \mathsf{dif}(h_i, h_j) = body(h_j) \setminus \mathsf{dif}(h_i, h_j)$. Let us further assume that there are two normal logic programs $\mathcal{P}_m = \{h_i, h_j\}$ and $\mathcal{P}_n = \{h_k\}$. An interpretation I is a model for $\mathcal{P}_m$ iff I is a model for $\mathcal{P}_n$.*

Proof. ($\rightarrow$) An interpretation I is a model for logic program $\mathcal{P}_m$, then by Definition 2.23 both h_i and h_j are mapped to *true* and what follows by Corollary 2.10, we have the following possibilities:

1. $head(h_i) \in I$ and $head(h_j) \in I$, then $head(h_k) \in I$ and therefore, by Corollary 2.10 I is a model for h_k, and further by Definition 2.23, I is a model for $\mathcal{P}_n$.
2. There is an atom a_z, where $a_z \in body(h_i)$ and $a_z \in body(h_j)$, and $a_z \notin I$. Therefore, by Definition 4.12 $a_z \notin \mathsf{dif}(h_i, h_j)$, and what follows $a_z \in body(h_k)$. Thus, by Corollary 2.10, I is a model for h_k and further by Definition 2.23, I is a model for $\mathcal{P}_n$.
3. There is a literal l_z, where $l_z \in body(h_i)$ and $l_z \in body(h_j)$, and $\mathsf{b}(l_z) \in I$. Therefore, by Definition 4.12 $l_z \notin \mathsf{dif}(h_i, h_j)$, and what follows $l_z \in body(h_k)$. Thus, by Corollary 2.10, I is a model for h_k and further by Definition 2.23, I is a model for $\mathcal{P}_n$.

($\leftarrow$) An interpretation I is a model for logic program $\mathcal{P}_n$, then by Definition 2.23 h_k is mapped to *true* and what follows by Corollary 2.10, we have the following possibilities:

1. $head(h_k) \in I$, then $head(h_i) \in I$ and $head(h_i) \in I$, and therefore, by Corollary 2.10 I is a model for h_i and h_j, and further by Definition 2.23, I is a model for $\mathcal{P}_m$.
2. There is an atom a_z, where $l_z \in body(h_k)$, and $a_z \notin I$. Since $body(h_k) \subset body(h_i)$ and $body(h_k) \subset body(h_j)$, then $a_z \in body(h_i)$ and $a_z \in body(h_j)$, and what follows from Corollary 2.10, I is a model for h_i and h_j, and further by Definition 2.23, I is a model for $\mathcal{P}_m$.
3. There is a literal l_z, where $l_z \in body(h_k)$, and $\mathsf{b}(l_z) \in I$. Since $body(h_k) \subset body(h_i)$ and $body(h_k) \subset body(h_j)$, then $l_z \in body(h_i)$ and $l_z \in body(h_j)$, and what follows from Corollary 2.10, I is a model for h_i and h_j, and further by Definition 2.23, I is a model for $\mathcal{P}_m$. $\square$

On the ground of the Proposition 4.2 we can define the second kind of logic programs simplification. If a given logic program $\mathcal{P}$ contains two complementary Horn clauses: h_i and h_j, then we can remove both clauses from program $\mathcal{P}$ and add a "new" one that has the same head as both former clauses and the body that is equal the body of either of the clauses h_i or h_j minimised by the symmetric difference between them.

Definition 4.10. Let $\mathcal{P}_i$ be a normal logic program and $\mathbb{P}$ be a family of logic programs. The mapping $\mathsf{simp2} : \mathbb{P} \longrightarrow \mathbb{P}$ is defined as follows:

$$\mathsf{simp2}(\mathcal{P}_i) = \begin{cases} \mathcal{P}'_i & \text{if there are } h_j, h_k \in \mathcal{P}_i \text{ that are complementary} \\ \mathcal{P}_i & \text{otherwise} \end{cases}$$

where $\mathcal{P}'_i = (\mathcal{P}_i \setminus \{h_j, h_k\}) \cup h_l$ and $head(h_l) = head(h_j) = head(h_k)$, $body(h_l) = body(h_j) \setminus \mathsf{dif}(h_j, h_k) = body(h_k) \setminus \mathsf{dif}(h_j, h_k)$.

The Figure 4.9 illustrates how both simplifications can be applied to a logic program. Programs $\mathcal{P}_2$ and $\mathcal{P}'_2$ were obtained from the program $\mathcal{P}_1$ by means of the simp1 while the program $\mathcal{P}''_2$ was obtained from the program $\mathcal{P}_1$ by means of the simp2. Afterwards, we can obtain the program $\mathcal{P}_3$ from either of the programs $\mathcal{P}_2$, $\mathcal{P}'_2$ or $\mathcal{P}''_2$ by means of the simp1.

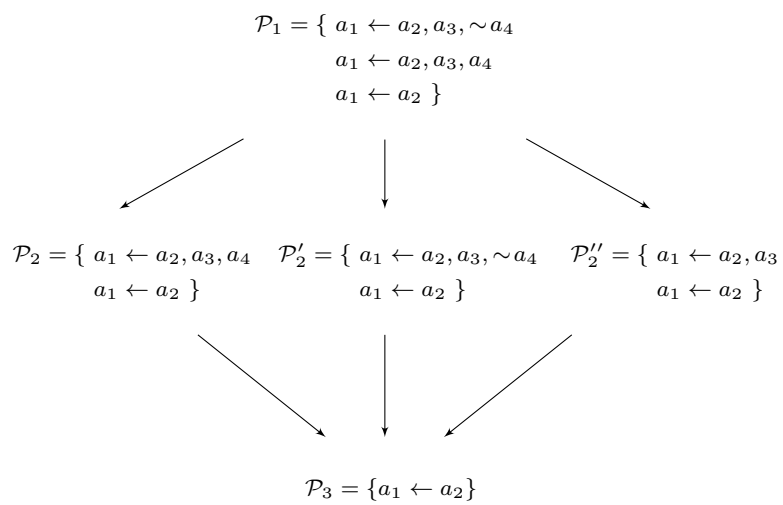

Fig. 4.9 Example of normal logic program simplification.

Combining both mappings that can reduce the number of Horn clauses in a given logic program, i.e. simp1 and simp2, we obtain the simplification mapping that returns a logic program that cannot be further simplified by means of simp1 or simp2.

Definition 4.11. Let $\mathcal{P}$ be a normal logic program and $\mathbb{P}$ the family of normal logic programs. The mapping $\mathsf{simp} : \mathbb{P} \longrightarrow \mathbb{P}$ is defined as follows:

$$\mathsf{simp}(\mathcal{P}) = \begin{cases} \mathsf{simp}(\mathsf{simp1}(\mathcal{P})) & \text{if } \mathsf{simp1}(\mathcal{P}) \neq \mathcal{P} \\ \mathsf{simp}(\mathsf{simp2}(\mathcal{P})) & \text{if } \mathsf{simp2}(\mathcal{P}) \neq \mathcal{P} \\ \mathcal{P} & \text{otherwise} \end{cases}$$

Summing up this subsection, in order to translate a neural network into a logic program we will implement the pedagogical approach, i.e. we will map all possible states of the input layer neurons into states of the output layer neurons, and afterwards, we will simplify obtained logic programs my means of the mappings simp1 (Definition 4.7) and simp2 (Definition 4.10).

Similar simplification methods were described by Garcez et al. [14, p. 131], where two definitions were given: Definition 5.2.5 concerning subsumption, what corresponds to the mapping simp1, and Definition 5.2.6 concerning complementary literals, what corresponds to the mapping simp2. The difference is that both simplification methods introduced by Garcez et al. are in the form of definitions without proofs for logic programs, because they arise from considerations concerning tehniques that allow to "trim" the search space for the pedagogical translation algorithm. Here, the simplification methods are proven to work regardless from the logic program origins.

4.2 Neural networks training

There are several possible ways of neural networks training. As we have assumed at the beginning, we are going to make use of the Backpropagation algorithm. Therefore, we need to specify the way the training examples are generated.

Looking back at the end of the Chapter 2 we see that a Horn clause h_i is an abductive goal $\mathcal{G}$ when a logic program $\mathcal{P}$ is concerned, if the head of the clause h_i does not belong to the least Herbrand model $M_{\mathcal{P}_p^{\mathsf{dn}(h_i)}}$, i.e. the least Herbrand model of the positive version of the program $\mathcal{P}$ modified w.r.t. the clause h_i with all negations converted to the upper index n. By means of the translation algorithm $T_{\mathcal{P}\to\mathfrak{N}}$ given in the Subsection 4.1.1 we obtain a neural network for the positive version of the logic program $\mathcal{P}$, which in turn is modified w.r.t. the Horn clause h_i with all negations occurring in its body changed to the upper index n, i.e. we obtain a neural network for the program $\mathcal{P}_p^{\mathsf{dn}(h_i)}$.

Additionally, we know that a neural network that was created by means of the translation algorithm $T_{\mathcal{P}\to\mathfrak{N}}$ models the way the immediate consequence operator $T_{\mathcal{P}}$ works for logic programs. In particular, we know that the operator $T_{\mathcal{P}}$ calculates the least Herbrand model for a definite logic program $\mathcal{P}$ when starting from an empty Herbrand interpretation $I_{\mathcal{P}}$—as the neural network does, where the empty interpretation is represented by the activation of all input layer neurons equal to -1 value.

Putting all things together, the neural network created by means of the translation algorithm $T_{\mathcal{P}\to\mathfrak{N}}$ for the logic program $\mathcal{P}_p^{\mathsf{dn}(h_i)}$ computes the least Herbrand model $M_{\mathcal{P}_p^{\mathsf{dn}(h_i)}}$, when starting from the point where all of the activations of the input layer neurons are set to -1 value, i.e. the counterpart for

the empty interpretation. Since the clause h_i is an abductive goal, concerning the logic program $\mathcal{P}$, we know that the output layer neuron associated with the atom that is the head of the clause h_i will not have activation greater that the $A_{\min}$ value, because otherwise the atom that is the head of the clause h_i would be considered *true* and what follows, it would be an element of the least Herbrand model $M_{\mathcal{P}_p^{\mathtt{dn}(h_i)}}$, i.e. the clause h_i would not be an abductive goal when the program $\mathcal{P}$ is considered.

For those reason we have decided to create one training example for the neural network, which is the following:

- activations of all of the input layer neurons are set to -1,
- activations of all of the output layer neurons are supposed to be equal 1.

This training example corresponds to the situation in which the immediate consequence operator $T_{\mathcal{P}}$ starts from the empty interpretation and is iterated till it reaches a fixpoint.

The condition of the activations of all of the output layer neurons to be equal 1 is a trade-off between the information we want to introduce to the neural network and the information we want it to obtain "by itself". We know that in order for the clause h_i to stop being an abductive goal its head has to be an element of the least Herbrand model $M_{\mathcal{P}_p^{\mathtt{dn}(h_i)}}$ computed by the neural network. Therefore, the output layer neuron associated with the head of the clause h_i should have activation that allows to consider it *true*. However, the question is what to do with the other output layer neurons. In order not to introduce to much our knowledge in the form of the training examples we decided to set a condition where all of the activations of the output layer neurons are equal 1.

One remark should be made here. If we set the training example as described in the former paragraph and apply the Backpropagation algorithm with recursion, then the trained neural network translated into a logic program most likely yields a logic program that consists only of facts. In order to avoid such situation we applied Backpropagation algorithm without recursion, i.e. after setting the activation of the input layer neurons we wait until the neural network stabilises itself and after we check the activation of the output layer neurons, we run Backpropagation algorithm only once from the output to input layer neurons, as if there were no recurrent connections.

4.3 The procedure

Now we can bring all the pieces together and describe the abductive procedure. The starting point is a knowledge base in the form of a logic program $\mathcal{P}$ and an abductive goal $\mathcal{G}$ in the form of a Horn clause h_i, which is not a logical consequence of the program $\mathcal{P}$. The first step is to swap possibly negated atoms in the program $\mathcal{P}$ and the Horn clause h_i for atoms with the same

number index, but instead of negation, with the upper index n. Hence we obtain positive version of the program $\mathcal{P}$, denoted by $\mathcal{P}_p$, and positive version of the Horn clause h_i, denoted by $\mathsf{dn}(h_i)$. Afterwards, the program $\mathcal{P}$ is modified w.r.t. the Horn clause h_i, i.e. atoms from the body of the Horn clause $\mathsf{dn}(h_i)$ are marked with the upper index h and added to the clauses from the logic program $\mathcal{P}_p$ in the form of facts—the newly obtained logic program is denoted by $\mathcal{P}_p^{\mathsf{dn}(h_i)}$. At this point we use the translation algorithm $T_{\mathcal{P}\to\mathfrak{N}}$ from logic programs to neural networks in order to translate program $\mathcal{P}_p^{\mathsf{dn}(h_i)}$ into a neural network. Our goal now is to train the neural network in such a way that the output layer neuron associated with the head of the clause h_i has its activation greater than the $A_{\min}$ value after the stabilisation of the neural network. In addition, the starting point for the neural network is the situation in which all of the input layer neurons have the minimal activation, i.e. -1. In this case the training method is the backpropagation algorithm. After the neural network reaches the required error level, it is translated back into a logic program by means of the translation algorithm $T_{\mathfrak{N}\to\mathcal{P}}$. The Horn clause h_i should be a logical consequence of the obtained logic program $\mathcal{P}'$, since its head belongs to the least Herband model of program $\mathcal{P}'$.

In order to obtain abductive hypotheses for a given logic program, $\mathcal{P}$ and abductive goal $\mathcal{G}$ we have to compare both logic programs, i.e. $\mathcal{P}$ and $\mathcal{P}'$. The easiest way is to calculate the symmetric difference between programs $\mathcal{P}$ and $\mathcal{P}'$ and consider it as an abductive hypothesis, because this difference is what makes the abductive goal obtainable from the knowledge base in the form of $\mathcal{P}$.

Definition 4.12. Let $\mathcal{P}_j$ and $\mathcal{P}_k$ be normal logic programs and h_i a Horn clause. The *symmetric difference* between $\mathcal{P}_j$ and $\mathcal{P}_k$ is defined as follows:

$$\mathsf{d}(\mathcal{P}_j, \mathcal{P}_k) =_{df} \{h_i \mid h_i \in \mathcal{P}_j \cup \mathcal{P}_k,\ h_i \notin \mathcal{P}_j \text{ or } h_i \notin \mathcal{P}_k\}$$

This solution returns two kinds of Horn clauses: those that were deleted from and those that were added to the logic program $\mathcal{P}$ during the neural network training. Such solution requires a strict definition which allows to state with certainty if a given Horn clause was added to the logic program or if it was just modified.

Another point is that there could be defined more sophisticated factors that allow to measure the similarity between two logic programs, e.g. the number of Horn clauses, facts, atoms in the bodies of the Horn clauses and so on, which could account for better classification of the obtained results. In Chapter 6 (see p. 119) I compare my method with few existing solutions and discuss differences between obtained abductive hypotheses as well as differences in the approach to the problem of the implementation of the abductive procedure. Therefore, for the purpose of this work I will stay with the symmetric difference between two logic programs as a definition of abductive hypothesis. However, I tackle those problems in the section devoted the future works.

4.3.1 Evaluation of abductive hypotheses

For every knowledge base and abductive goal there are many different abductive hypotheses [58, p. 22]. We can evaluate them according to multiple criteria and identify those that are "better" in some sense. There are two properties of abductive hypotheses that are intuitive and come into the foreground, namely the consistency with the knowledge base and the inability of deriving abductive goal from the hypothesis alone (see for example [1, pp. 74–75]). Another one is the simplicity of the abductive hypotheses, which may be construed in different ways, for instance syntactically, as the length of the hypothesis, like in [6], or proof-theoretically, as inability to derive one abductive hypothesis from another, like in [28] (see also the Chapter 6 on p. 119).

There are also more sophisticated methods of abductive hypotheses evaluation, like for example in the implementation of abductive reasoning given in [30] (the procedure is described in the Chapter 6). Komosiński et al. [30] distinguish five criteria according to which a multi-criteria analysis can be conveyed in order to classify abductive hypotheses and obtain those that are non-dominated concerning those criteria, i.e. the "best ones":

1. Consistency.
2. Significance.
3. Complexity.
4. Operational complexity.
5. Overlapping.

Along with those criteria there were also described ways of calculating them as a numerical values for the further analysis. The abductive hypothesis can receive two values as far as the consistency criterion is concerned: 0 when the abductive hypothesis is inconsistent with the knowledge base, and 1 when the abductive hypothesis is consistent with the knowledge base. The second criterion, which is the significance also can return two values for an abductive hypothesis: 0 when the abductive goal is obtainable from the hypothesis alone, and 1 when the abductive goal is not obtainable from the hypothesis alone. In order to automatise the process of calculation both criteria for a given abductive hypothesis Komosiński et al. [30] implemented and used the analytic tableau method. Complexity is measured as the number of distinct variables in the abductive hypothesis h. Operational complexity is mesured as the number of two-argument logical operators that are present in the abductive hypothesis h. As authors write in [30], criteria three and four do not necessarily overlap, i.e. there are abductive hypotheses for which they yield different results. Finally, overlapping is measured as the number of variables that are occur in both: the abductive goal and the abductive hypothesis. Having calculated all values for every obtained abductive hypothesis we can run multi-criteria analysis, where the first two criteria are preferred to be maximised, and the remaining three—to be minimised. In such a way

Komosiński et al. constructed a fully automated procedure that allows to evaluate abductive hypotheses on multiple levels.

The abductive procedure presented in this work enforces the first mentioned property, i.e. the consistency with the knowledge base. The reason is that it is impossible to obtain hypotheses of the form of Horn clauses with negated atoms in their heads. In case of the second property there were two steps which facilitate generation of abductive hypotheses that are not too strong. The first step is the following: in the translation algorithm from logic programs to neural networks (see the Subsection 4.1.1.1) the output layer neuron that is associated with the atom that is the head of the abductive goal is not connected with an additional connection with the hidden layer truth neuron, therefore it cannot receive directly the minimal input to be concerned *true* and would not be translated into a fact that easy. The second step concerns the prohibition of creating additional connections between the input layer neurons and the additional hidden layer neurons that are in turn connected with the output layer neurons which are associated with the same atoms as the considered input layer neurons, therefore there cannot establish simple self sustaining loops that could lead to the formulation of facts after the translation of the neural network. However, as it turns out in the Chapter 5 devoted for the description of the implementation of the abductive procedure and obtained results, those steps do not fully prevent formation of facts with the head that is the abductive goal.

4.4 Chapter summary

This Chapter is a direct continuation of the results obtained in the first Chapter (see p. 11). Let us recall the Corollary 2.15 that given a normal logic program $\mathcal{P}$ and a Horn clause h_i, if $\mathcal{P} \nvDash h_i$, then $head(h_i) \notin M_{\mathcal{P}_p^{\mathsf{dn}(h_i)}}$, i.e. if the clause h_i is not a consequence of the program $\mathcal{P}$, then its head does not belong to the least Herbrand model of the positive version of the program $\mathcal{P}$ modified w.r.t. the clause h_i, where all negations were swapped for the upper index n. This is the starting point for the algorithm $T_{\mathcal{P}\to\mathfrak{N}}$ that translates a logic program into a neural network, because we want to translate the program $\mathcal{P}_p^{\mathsf{dn}(h_i)}$ into a neural network.

The translation algorithm $T_{\mathcal{P}\to\mathfrak{N}}$, which is based on the work of Garcez et al. [14, ch. 5] described in details in the Subsection 4.1.1.1, is introduced in the Subsection 4.1.1.2. The modifications that were introduced regarded the addition of the additional hidden layer neurons and additional connections:

- for every output layer neuron, with the exception of the facts, there is added a number of additional hidden layer neurons,
- the additional connections were established between the input and hidden layer neurons, and the hidden and output layer neurons.

There were also introduced some restrictions that prevent from establishing additional connections that after the translation of the neural network into a logic program could lead to the formulation of tautologies of the form $a_1 \leftarrow a_1$, and those which would connect the head of the Horn clause that is an abductive goal with the *truth* neuron.

The first kind of problems that had to be solved at this stage of the abductive procedure was caused by the introduced modifications and concerned the adjustment of the equations that allow to calculate basic properties of the constructed neural network, such as the weight of the connections W, the value $A_{\min}$ or the biases for the hidden and output layer neurons.

The problem of the second kind concerned the fact that the immediate consequence operator $T_{\mathcal{P}}$ does not differentiate between atoms and their counterparts with the upper index h, i.e. atoms that appear in the form of the heads of facts in a logic program after its modification w.r.t. a Horn clause. The solution involved introduction of neurons with the activation function that was dependent from the $A_{\min}$ value in the layer that is located between the output and input layer neurons.

At the end of the Subsection 4.1.1.2 the set of factors was defined that can influence the process of the abductive hypotheses generation during the training of the neural network. Those factors concerned mainly the architecture of the neural network.

Afterwards, in the Subsection 4.1.2 the algorithm $T_{\mathfrak{N}\rightarrow\mathcal{P}}$ that translates given neural network into a logic program. The employed approach here is the pedagogical one, i.e. all possible states of the input layer neurons are mapped into states of the output layer neurons. Despite the disadvantage of the pedagogical approaches, which is their complexity, this is the only approach that guarantees the soundness and the completeness of the translation algorithm. In order to reduce the number of the Horn clauses obtained during the translation process two kinds of simplification mappings for the logic programs were introduced, which combined result in a mapping that produces a logic program that cannot be further simplified. Similar solutions can be found in the work of Garcez et al. [14], although here in this work the properties of those simplifications are proven on the ground of the logic programs, while in the work of Garcez et al. they are a consequence of the methods that reduce the number of the search space for the translation algorithm.

The translation algorithm from neural networks to logic programs closes the Section 4.1 and we move on to the Section 4.2, where the problem of neural networks training is concerned. The chosen training strategy is the following:

- the training set consists only of one training example, where the initial input for every neuron from the input layer is set to -1 and the expected activation of every neuron from the output layer is 1,
- the Backpropagation algorithm is used without recursion and the state of the output layer neurons is checked after the stabilisation of the neural network.

The arguments for such approach that are given in the Section 4.2 involve the method of acquiring the least Herbrand model for a given logic program by means of the $T_{\mathcal{P}}$ operator and the balance between the knowledge we want to introduce to the neural network in the form of the training examples and the knowledge we want the neural network to obtain "alone" during the training process.

Finally, in the Section 4.3 the whole abductive procedure is described along with the last step, i.e. the comparison between two logic programs—the initial one and the one that results from the translation of the trained neural network. In this step the abductive hypotheses are obtained in the form of a symmetric difference between the initial logic program $\mathcal{P}$ and the logic program $\mathcal{P}'$ that results from the translation of the trained neural network. At the end of the Section 4.3 I briefly discuss properties of abductive hypotheses.

Now we move to the presentation of obtained results from the implementation of the whole abductive procedure. The translation algorithms and the neural network training process were implemented in the Framsticks [32] environment, while the simplification of the logic programs was done by means of the scripts written in Python. Obtained results show that this is not always the case that for a given logic program and an abductive goal the abductive hypotheses are the same, even if the factors for the neural network creation remain unchanged. However, there are some regularities that can be found in the form of the most common abductive hypotheses across different neural networks creation conditions.

Chapter 5
Implementation and results

In the Chapter 2 (see p. 11) formal grounds for the definition of abductive goal in the form of a Horn clause were introduced in terms of the immediate consequence operator $T_{\mathcal{P}}$, where the knowledge base is formalised as a normal logic program. Afterwards, in the Chapter 4 (see p. 65), those definitions served as the basis for the modifications introduced to the Garcez's et al. [14] translation algorithm from logic program to neural networks, as well as for the neural networks training strategy, in order to be able to obtain abductive hypotheses by means of the neural network training. In this Chapter the implementation of the abductive procedure is introduced along with obtained results, which is a joint work described in [13].

There are many different platforms and software libraries that enable implementation of artificial neural networks. However, in many cases there are limitations that concern for example kinds of neurons used or the neural networks structures. Therefore, in the first section tools used for the implementation of the abductive procedure are described along with a brief justification for the choices. In the following section three examples of abductive problems are introduced in order to follow step by step through the implementation of the abductive procedure. At the end of each example obtained results are discussed. The Chapter ends with a summary and a description of the future work.

5.1 Implementation

Framsticks [32] software was chosen for neural networks implementation. The reasons that support this decision are the following: Framsticks environment gives a user full control on architecture of created neural networks, i.e. custom neurons and arbitrary neural networks constructions are fully supported; possibility of performing computational experiments concerning artificial neural networks, i.e. there is available an advanced neural networks simulator and

a script language that enables to create any kind of experiments. In addition, Framsticks software was already used in experiments that concerned logical abduction [30, 31] (see more in the next chapter). The implementation of translation algorithms from logic programs to neural networks ($T_{\mathcal{P}\to\mathfrak{N}}$) and reverse, from neural networks to logic programs ($T_{\mathfrak{N}\to\mathcal{P}}$), is also conveyed in the Framsticks environment, since aforementioned script language is already available for the user.

In Subsection 4.1.2 I described the translation algorithm from neural networks to logic programs that returns all combinations of activations of the input layer neurons that make a given output layer neuron activation above the $A_{\min}$ value. Every combination is saved as a separate Horn clause, where the atom associated with the neuron from the output layer forms the head of the Horn clause, and depending on the activation below or above $A_{\min}$ value atoms or their negations, associated with neurons from the input layer are ascribed to the body of the Horn clause. Therefore, for each head we can create a complete table that contains all possible combinations of the atoms from the body denoted as 1, when atom is not preceded by the negation, and 0, when atom is preceded by the negation, and where the head has 1 value, if given combination excites the associated output layer neuron above $A_{\min}$ value or 0 otherwise. For example, such table for atom a_1 from the logic program from example 4.9:

$$\begin{aligned}\mathcal{P}_1 = \{\ & a_1 \leftarrow a_2, a_3, \sim a_4 \\ & a_1 \leftarrow a_2, a_3, a_4 \\ & a_1 \leftarrow a_2\ \}\end{aligned}$$

looks as it is shown in Table 5.1, where clause $a_1 \leftarrow a_2, a_3, \sim a_4$ is represented as row 7, clause $a_1 \leftarrow a_2, a_3, a_4$ is represented as row 8, and clause $a_1 \leftarrow a_2$ is represented by rows 5–8. The reason why the Horn clause $a_1 \leftarrow a_2$ is represented by four rows is that the neuron from the input layer associated with atom a_2 causes the output layer neuron that is associated with atom a_1 to be activated above the $A_{\min}$ value regardless of the activation of the input layer neurons associated with atoms a_3 and a_4.

The simplification of logic programs is conveyed as a reduction of Boolean functions by means of Quine and McCluskey algorithm [36, 42] for a given set of Horn clauses with the same head. In the example shown in Table 5.1 the Quine and McCluskey algorithm returns the following Horn clause: $a_1 \leftarrow a_2$, which is exactly the same as the one obtained by means of the simplification methods. This part of the procedure was implemented in Python programming language by means of official packages for the Quine and McCluskey algorithm implementation.

Both modules, i.e. the scripts written in Framsticks and the simplification method written in Python, were managed by means of the script written in the Linux shell language.

Table 5.1 Table with the information which configurations of the input layer neurons associated with atoms a_2, a_3 and a_4 cause the output layer neuron, that is associated with atom a_1, to have activation greater than the $A_{\min}$ value.

	a_1	a_2	a_3	a_4
1	0	0	0	0
2	0	0	0	1
3	0	0	1	0
4	0	0	1	1
5	1	1	0	0
6	1	1	0	1
7	1	1	1	0
8	1	1	1	1

5.2 Results

The first two examples concern generation of abductive hypotheses for abductive goals in the form of facts. The procedure is equivalent to the situation when we want to generate abductive hypotheses in the form of atoms, as it was mentioned at the end of Chapter 2. This detail is going to be used while discussing differences between the abductive procedure described in this work and the Abductive Logic Programming approach (see Section 6.1 on p. 119).

Let us consider the two following examples of logic programs $\mathcal{P}_1$ and $\mathcal{P}_2$ [13]:

$$\begin{aligned} \mathcal{P}_1 = \{ \quad & a_1 \leftarrow a_2, a_3 \\ & a_2 \leftarrow a_3 \\ & a_2 \leftarrow a_1 \quad \} \end{aligned} \tag{5.1}$$

$$\begin{aligned} \mathcal{P}_2 = \{ \quad & a_1 \leftarrow a_2 \\ & a_1 \leftarrow a_3 \\ & a_4 \leftarrow a_5 \\ & a_3 \leftarrow a_5 \\ & a_2 \leftarrow a_3 \quad \} \end{aligned} \tag{5.2}$$

and abductive goal in the form of a fact:

$$\mathcal{G} = \mathcal{G}_{\mathcal{P}_1} = \mathcal{G}_{\mathcal{P}_2} : a_1 \leftarrow \tag{5.3}$$

As we can see, the abductive goal is the same for both logic programs, therefore I will refer to it as simply $\mathcal{G}$.

Since $body(\mathcal{G}) = \emptyset$ and there are no negations in both logic programs $\mathcal{P}_1$ and $\mathcal{P}_2$, then modified positive versions of logic programs $\mathcal{P}_1$ and $\mathcal{P}_2$ are the same as the non-modified logic programs $\mathcal{P}_1$ and $\mathcal{P}_2$:

$$\mathcal{P}_{1p}^{\mathsf{dn}(\mathcal{G})} = \mathcal{P}_1$$
$$\mathcal{P}_{2p}^{\mathsf{dn}(\mathcal{G})} = \mathcal{P}_2$$

The methodology used to generate abductive hypotheses for the logic program $\mathcal{P}_1$ and $\mathcal{P}_2$ is the same, therefore I am going to show the neural network and the experiment setup only for the logic program $\mathcal{P}_1$ and for the logic program $\mathcal{P}_2$ we will only discuss obtained results.

The neural network that represents logic program $\mathcal{P}_1$ and abductive goal $\mathcal{G}$ is depicted in the Figure 5.1. The association between neurons and atoms is the following:

$$\mathfrak{i}_1 \leftrightsquigarrow a_1 \qquad \mathfrak{h}_1 \leftrightsquigarrow h_3 \qquad \mathfrak{o}_1 \leftrightsquigarrow a_2$$
$$\mathfrak{i}_2 \leftrightsquigarrow a_2 \qquad \mathfrak{h}_2 \leftrightsquigarrow h_2 \qquad \mathfrak{o}_2 \leftrightsquigarrow a_1$$
$$\mathfrak{i}_3 \leftrightsquigarrow a_3 \qquad \mathfrak{h}_3 \leftrightsquigarrow h_1 \qquad \mathfrak{o}_3 \leftrightsquigarrow a_3$$

where:

$$h_1 : a_1 \leftarrow a_2, a_3$$
$$h_2 : a_2 \leftarrow a_3$$
$$h_3 : a_2 \leftarrow a_1$$

As I have mentioned in Section 4.1.1, there are several factors that have to be established and which influence the process of a neural network training. One of those factors is the number of additional hidden layer neurons and for that reason in the Figure 5.1 there are three groups of additional hidden neurons but not a specific number of them:

- $\mathfrak{h}_4, \ldots, \mathfrak{h}_i$ are generated for the output layer neuron $\mathfrak{o}_1$,
- $\mathfrak{h}_j, \ldots, \mathfrak{h}_k$ are generated for the output layer neuron $\mathfrak{o}_2$,
- $\mathfrak{h}_l, \ldots, \mathfrak{h}_m$ are generated for the output layer neuron $\mathfrak{o}_3$.

The number of additional hidden layer neurons per output layer neuron is not specified here, because we have generated multiple neural networks with different number of additional hidden layer neurons for both problems. However the overall structure of the neural network is the same in each case.

In the Figure 5.2 the exemplary neural network from the Figure 5.1 is visible as implemented in Framsticks environment. In this case the neural network has one additional hidden layer neuron per output layer neuron (black neurons in the hidden layer).

We have designed six different experimental conditions listed in the Table 5.2. For each experimental setup we have generated 100 neural networks and trained them with one training example, as it is described in the Section 4.2.

After the translation from neural network to logic program and simplification we have obtained different programs for different experimental setups, or sometimes even within one experimental setup. Let us consider the logic

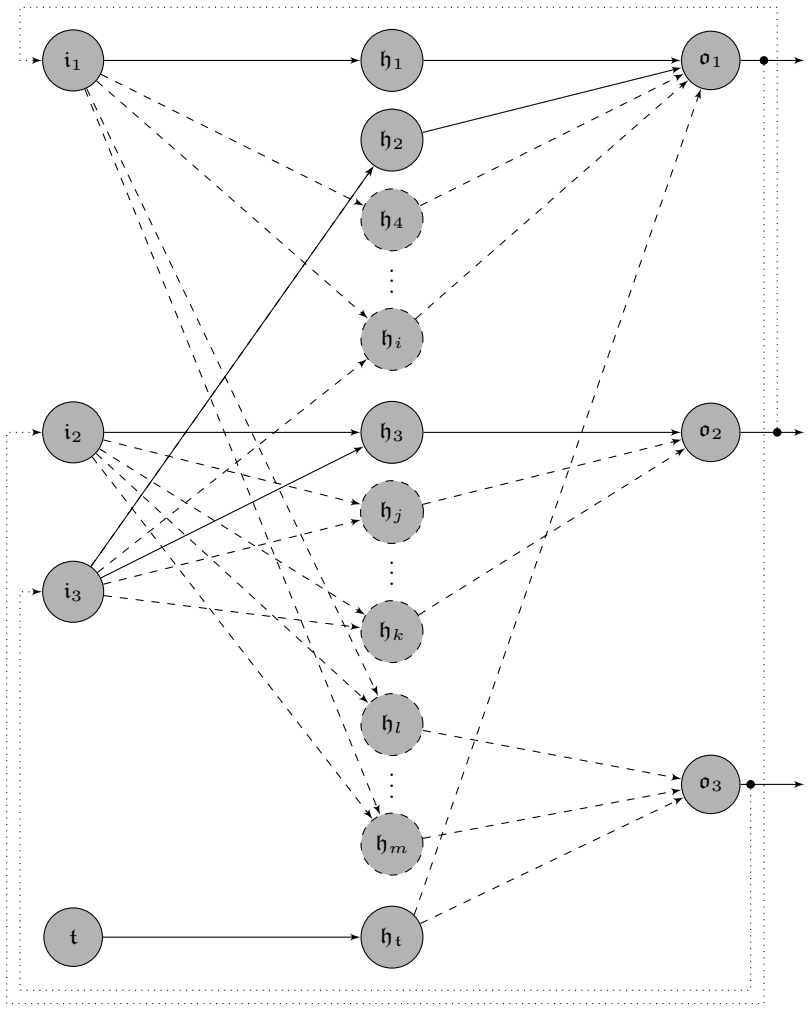

Fig. 5.1 Neural network for the logic program $\mathcal{P}_1$ and abductive goal $\mathcal{G}$. Arrows stand for connections in the modified logic program $\mathcal{P}_1$ w.r.t. the goal $\mathcal{G}$, dashed arrows stand for additional connections and dotted arrows for recurrent connections.

program $\mathcal{P}_1$ and the abductive goal $\mathcal{G}$. After the translation and simplification of the trained neural network the most common logic program we have obtained was the following:

$$\begin{aligned} \mathcal{P}_1' = \{ \quad & a_1 \leftarrow a_2, a_3 \\ & a_2 \leftarrow a_3 \\ & a_2 \leftarrow a_1 \\ & a_3 \leftarrow \quad \} \end{aligned} \tag{5.4}$$

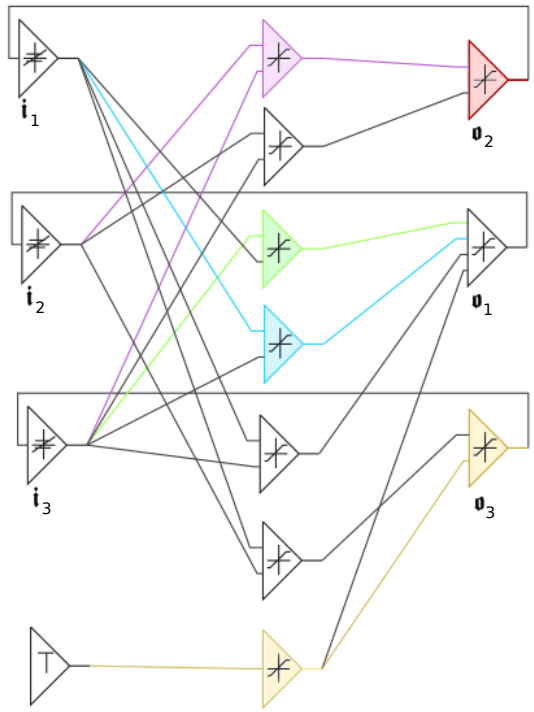

Fig. 5.2 The neural network from Figure 5.1 as seen in the Framsticks environment.

Table 5.2 Experimental conditions.

No	Additional hidden layer neurons per output layer neuron	Bias for each additional hidden layer neuron
1	1	0.0
2	17	0.0
3	1	3.3
4	17	3.3
5	1	6.6
6	17	6.6

Therefore, according to the Definition 4.12 the difference between the initial logic program $\mathcal{P}_1$ and the program $\mathcal{P}_1'$, obtained as a result of the abductive procedure, is the fact $a_3 \leftarrow$. It is a "good" abductive hypothesis taking into account its minimality, consistency with the knowledge base and the inability of deriving the abductive goal $\mathcal{G}$ only from the abductive hypothesis alone (more about a characteristics of desired properties of abductive hypotheses was described in the Subsection 4.3.1 on p. 100).

However, the program $\mathcal{P}_1'$ was not the only one obtained during the execution of abductive procedure. Figures 5.3 and 5.4 present results obtained for the logic programs $\mathcal{P}_1$ and $\mathcal{P}_2$, respectively. The numbers in the tuples on the horizontal axis describe the experimental condition: the first number is the bias for the additional hidden layer neurons and the second is the number of those neurons per output layer neuron. The vertical axis stands for the number of trials—for each experimental condition there were 100 trials of the abductive procedure, but within one experimental setup we could get multiple results. For example, when we consider the diagram with obtained logic programs for the logic program $\mathcal{P}_1$ and abductive goal $\mathcal{G}$, depicted in the Figure 5.3, we can see that for the bias for additional hidden layer neurons set to 0.0—regardless of the number of additional hidden layer neurons—there is only one logic program obtained for all trials, namely "blue one". Within the same diagram, moving to the experimental condition where the bias for additional hidden layer neurons is set to 6.6 and their number to 17, we see that there are two logic programs obtained: "orange one" in about 20 trials and "green one" in the rest.

In case of the program $\mathcal{P}_1$ there are four different logic programs obtained considering all experimental conditions, and each logic program is represented as specific colour. For example, the logic program $\mathcal{P}_1'$ given in 5.4 is the most common obtained result (i.e. for conditions where the bias for additional hidden layer neurons was set to 0.0 and 3.3) and is marked in the Figure 5.3 by the blue colour. The rest of obtained programs is listed below, where the program $\mathcal{P}_1''$ is the second most common, especially when conditions with the bias for additional hidden layer neurons set to 6.6 are concerned (in the Figure 5.3 program $\mathcal{P}_1''$ is marked with the green colour, $\mathcal{P}_1'''$ with the orange colour and $\mathcal{P}_1''''$ with the yellow colour).

$$\mathcal{P}_1'' = \{ \begin{array}{l} a_1 \leftarrow \\ a_2 \leftarrow \\ a_3 \leftarrow \end{array} \} \qquad \mathcal{P}_1''' = \{ \begin{array}{l} a_1 \leftarrow \\ a_2 \leftarrow a_1 \\ a_2 \leftarrow a_3 \\ a_3 \leftarrow \end{array} \} \qquad \mathcal{P}_1'''' = \{ \begin{array}{l} a_1 \leftarrow a_3 \\ a_1 \leftarrow a_2 \\ a_2 \leftarrow a_1 \\ a_2 \leftarrow a_3 \\ a_3 \leftarrow \end{array} \}$$

What can be seen here is that every obtained logic program contains the fact $a_3 \leftarrow$, which is a "good" abductive hypothesis, i.e. it is consistent with the knowledge base, the abductive goal is not obtainable from the hypothesis alone, and it is minimal in the sense that there is no other abductive hypothesis from which it could be obtained. Now, the reason the program $\mathcal{P}_1'$ is "better" from the program $\mathcal{P}_1''$ is that it does not remove any information from the knowledge base (what in this particular case is good). In addition, although the program $\mathcal{P}_1''$ contains the fact $a_3 \leftarrow$, it contains also the fact $a_1 \leftarrow$, which is not desired, because it is the abductive goal $\mathcal{G}$. The process

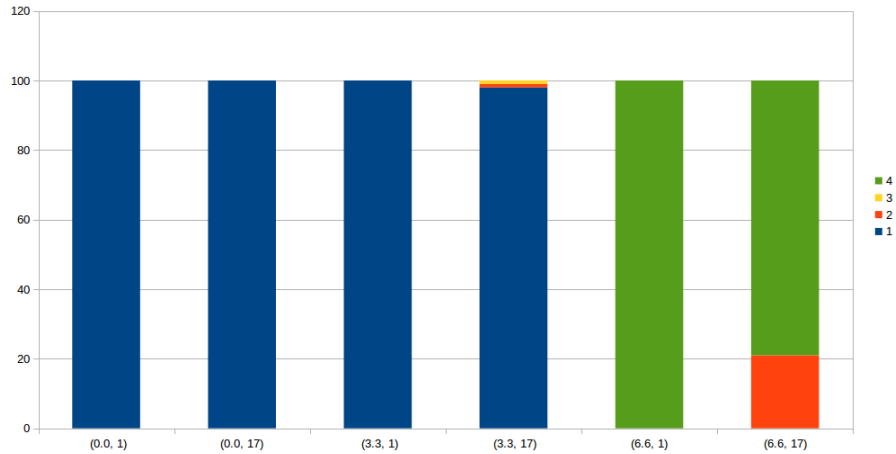

Fig. 5.3 Results obtained for the logic program $\mathcal{P}_1$ and the abductive goal $\mathcal{G}$. Each colour represents obtained logic program at the end of the abductive procedure.

of establishing which abductive hypothesis is better can be fully automated, like it is done for example in [30].

Let us now move to the second abductive problem. In the case of the logic program $\mathcal{P}_2$ the most common result of the abductive procedure application is the logic program $\mathcal{P}_2'$:

$$\begin{aligned} \mathcal{P}_2' = \{ \quad & a_1 \leftarrow a_2 \\ & a_1 \leftarrow a_3 \\ & a_4 \leftarrow \\ & a_3 \leftarrow \\ & a_2 \leftarrow \\ & a_5 \leftarrow \quad \} \end{aligned} \tag{5.5}$$

In this case the abductive hypothesis that is "good" is a fact $a_5 \leftarrow$. It allows to obtain the abgictive goal $\mathcal{G}$ when added to the llogic program, is consistent with the knowledge base, and cannot be derived from any other abductive hypothesis. Although the program $\mathcal{P}_2'$ contains the fact $a_5 \leftarrow$ it also removes 3 Horn cluses from the initial knowedge base $\mathcal{P}_2$ and insted inserts 3 facts with the sema heads:

- $a_4 \leftarrow a_5$ removed, $a_4 \leftarrow$ added,
- $a_3 \leftarrow a_5$ removed, $a_3 \leftarrow$ added,
- $a_2 \leftarrow a_3$ removed, $a_2 \leftarrow$ added,
- $a_5 \leftarrow$ added.

However, this is the most common result when the bias of the additional hidden layer neurons is set to 0.0 or 6.6. Overall, there are 40 different logic

programs obtained during the experiment (see Appendix on p. 149), as it is shown in the Figure 5.4, where $\mathcal{P}_2'$ is marked by the blue colour. What is more, for the experimental condition with 17 additional hidden layer neurons and the bias for them set to 3.3 there is a rich diversity of obtained logic programs and it is hard to point out the one that dominates the rest of the results.

Next two common results were the following logic programs:

$$\mathcal{P}_2'' = \{ \begin{array}{l} a_1 \leftarrow a_2 \\ a_1 \leftarrow a_3 \\ a_4 \leftarrow a_5 \\ a_3 \leftarrow a_5 \\ a_2 \leftarrow a_3 \\ a_5 \leftarrow \quad \} \end{array} \tag{5.6}$$

$$\mathcal{P}_2''' = \{ \begin{array}{l} a_1 \leftarrow \\ a_4 \leftarrow \\ a_3 \leftarrow \\ a_2 \leftarrow \\ a_5 \leftarrow \quad \} \end{array} \tag{5.7}$$

As we can see, in this case the difference between obtained results is the number of Horn clauses exchanged for facts, i.e. the symmetric difference between the initial logic program $\mathcal{P}_2$ and the program $\mathcal{P}_2'$ is just the fact $a_5 \leftarrow$, what makes this program the best when the properties of abductive hypothesis are concerned. It is marked with the orange colour in the Figure 5.4. We can see tht this is the most common logic program obtained for conditions where the number of additional hidden layer neurons per output layer neuron was 1, and their bias was set to 3.3.

Moving further, in case of the logic program $\mathcal{P}_2'''$ all of the Horn clauses from the initial program $\mathcal{P}_2$ are removed and replaced by facts. In this case the abductive hypothesis has the property that we try to avoid, i.e. it contains the fact $a_1 \leftarrow$, which basically is the abductive goal $\mathcal{G}$. This logic program was obtained for the conditions where there were 17 additional hidden layer neurons per output layer neuron and their bias was set to 6.6.

In this scenario the most common obtained result turned out to be not the best abductive hypothesis in terms of the simplicity, while in the same time it has the other two desired properties, i.e. the resut is consistent with the knowledge base and the abductive goal is not derivable from th hypothesis alone. However, for one experimental result "the best" abductive hypothesis was the most common result obtained. In addition, despite the diversity of obtained logic programs through the abductive procedure for different experimental conditions, every logic program amongst 40 (see Figure 5.4) contains the fact $a_5 \leftarrow$, which is a "good" abductive hypothesis.

The two considered examples were based on the scenario, where the abductive goal is of the form of a fact, which is equivalent to the situation where the abductive goal is of the form of an atom. Let us now consider an example, where the abductive goal is of the form of a Horn clause. The initial knowledge base is of the following form of the logic program $\mathcal{P}_2$:

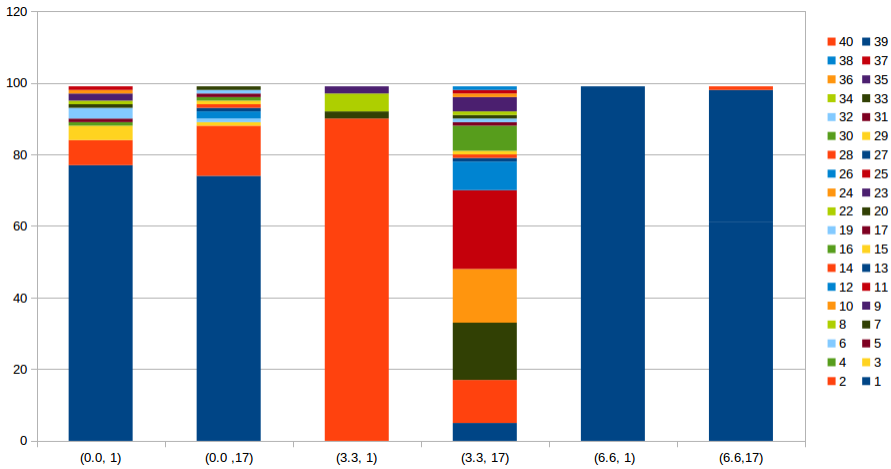

Fig. 5.4 Results obtained for the logic program $\mathcal{P}_2$ and the abductive goal $\mathcal{G}$. Each colour represents obtained logic program at the end of the abductive procedure.

$$\begin{aligned}\mathcal{P}_2 = \{ \quad & a_1 \leftarrow a_2 \\ & a_1 \leftarrow a_3 \\ & a_4 \leftarrow a_5 \\ & a_3 \leftarrow a_5 \\ & a_2 \leftarrow a_3 \quad \}\end{aligned}$$

and the abdictive goal is the following Horn clause:

$$\mathcal{G}'_{\mathcal{P}_2} : a_4 \leftarrow a_2 \tag{5.8}$$

The most common logic programs obtained as a result of applying the abductive procedure were the following, respectively (in the Figure 5.5 the program $\mathcal{P}'_2$ is marked by the green colour, the program $\mathcal{P}''_2$ is marked by the blue colour, and the program $\mathcal{P}'''_2$ is marked by the orange colour):

$$\begin{aligned}\mathcal{P}'_2 = \{ \quad & a_1 \leftarrow a_2 \\ & a_1 \leftarrow a_3 \\ & a_4 \leftarrow \\ & a_3 \leftarrow \\ & a_2 \leftarrow \\ & a_5 \leftarrow \quad \}\end{aligned} \qquad \begin{aligned}\mathcal{P}''_2 = \{ \quad & a_1 \leftarrow a_2 \\ & a_1 \leftarrow a_3 \\ & a_4 \leftarrow a_1 \\ & a_3 \leftarrow \\ & a_2 \leftarrow \\ & a_5 \leftarrow \quad \}\end{aligned} \qquad \begin{aligned}\mathcal{P}'''_2 = \{ \quad & a_1 \leftarrow a_2 \\ & a_1 \leftarrow a_3 \\ & a_4 \leftarrow a_5 \\ & a_3 \leftarrow \\ & a_2 \leftarrow \\ & a_5 \leftarrow \quad \}\end{aligned}$$

Let us take for example the symmetric difference between the initial logic program $\mathcal{P}_2$ and the program $\mathcal{P}'''_2$, which is the most conservative, i.e. contains less changes to the initial knowledge base than the other two logic programs:

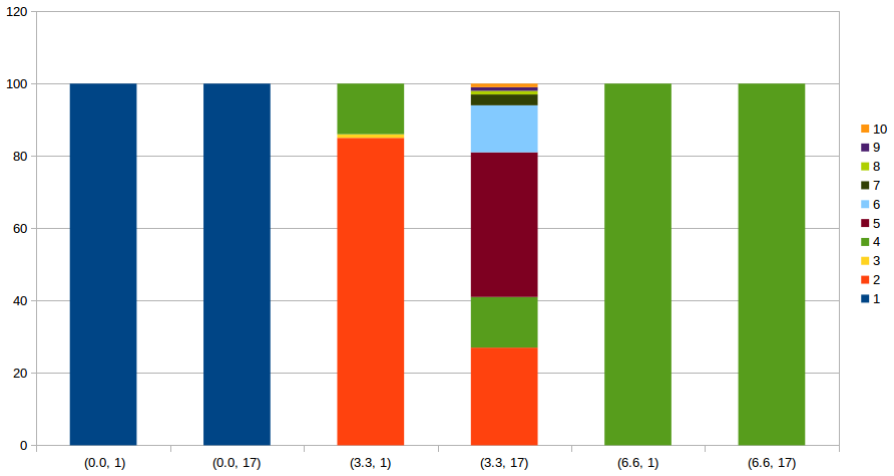

Fig. 5.5 Results obtained for the logic program $\mathcal{P}_2$ and the abductive goal $\mathcal{G}'_{\mathcal{P}_2}$. Each colour represents obtained logic program at the end of the abductive procedure.

- clauses removed: $a_3 \leftarrow a_5$, $a_2 \leftarrow a_3$,
- clauses added: $a_3 \leftarrow$, $a_2 \leftarrow$, $a_5 \leftarrow$.

As we can see, clauses $a_3 \leftarrow a_5$ and $a_2 \leftarrow a_3$ were stripped of their bodies. In addition a fact $a_5 \leftarrow$ was added. While all hypotheses obtained from those three logic programs are consistent with the knowledge base only hipotheses from programs $\mathcal{P}_2''$ and $\mathcal{P}_2'''$ have the property of being not to strong, i.e. the abguctive goal cannot be derived from those hypotheses alone.

The abductive goal is obtainable from all three the logic programs, therefore we can say that the goal was reached. However, we wouold like to obtain some connection between the body and the head of the abductive goal through abducive hypotheses and clauses from the initial knowledge base. Therefore, the logic program $\mathcal{P}_2''$ is candidate, since there is a connection from the knowledge base between the atom a_2 and a_1, and as a hypothesis a connection between the atom a_2 and a_1 was established, as well as making the atom a_2 as a fact. Unfortunately, this is not the case in the remaining two logic programs. In their cases the abductive goal is derivable from the knowledge base together with abductive hypotheses only because the head of the abductive goal is linked with one of the facts.

All results for this example are shown in the Figure 5.5. Overall, there were 10 (see the Appendix on p. 149) different logic programs that were obtained across all trials and experimental conditions.

5.3 Chapter summary

In the Section 4.3 (see p. 98) I have described in details particular steps in the abductive procedure and how they are combined together. At the beginning of this Chapter, in the Section 5.1, I have shown how individual modules of the abductive procedure were implemented and gave justification for the choices of used resources. The main part of the whole procedure, i.e. translation algorithms, neural networks and the backpropagation training algorithm, were implemented by means of the Framsticks [32] environment. The rest of the procedure, which include the simplification of logic programs by means of the Quine and McCluskey algorithm and automation of the execution of the all scripts, was executed by means of the Python programming language and Linux shell script.

Afterwards we have moved to the Section 5.2 devoted for the description of results obtained by means of the implemented abductive procedure. There were three cases considered based on two different logic programs, where the first two situations concerned abductive goals in the form of facts, and the third one concerned abductive goal in the form of a Horn clause with a body that is not empty. There are several factors that influence the way a neural network works when it is created by means of the translation algorithm from a logic program (see the Subsection 4.1.2 on p. 90). In experiments described in this Chapter the manipulation concerned only two of the factors: the number of additional hidden layer neurons per output layer neuron, and their bias. Obtained results indicate that those two factors influence the formulation of abductive hypotheses, i.e. logic programs that were results of the translation of the trained neural networks can be different for distinct experimental setups. For example, in experiment concerning the second logic program and the abductive goal in the form of a fact the total number of acquired different logic programs after the translation of the trained neural networks across all experimental conditions and trials was 40. In addition, for some initial settings we have obtained different results for different trials. We can describe it as certain "instability" of the abductive procedure for particular settings of the factors for the translation algorithm from neural networks to logic programs. However, there were also common properties of abductive hypotheses that held for all results obtained from a given experiment. Those common properties indicate that it is possible to obtain "good" abductive hypotheses by means of described abductive procedure, but it requires more wide-scale experiments concerning different abductive problems and the influence of the whole range of factors on the shape of neural networks obtained by means of the translation algorithm.

This brings us to the subject of future works. The plans are that the presented methodology will be used to convey experiments with more diverse types of logic programs and abductive goals. In addition, in order to check the influence of the combination of all factors that are considered in the translation algorithm form neural networks to logic programs on the creation

of abductive hypotheses, we would need to calculate much more examples for a given set of factors and determine smaller, reasonable intervals between values of those factors across experimental conditions. Results obtained from such experiments will have to analysed by means of tools such as the multi-criteria analysis.

Another point is that the way the neural networks were trained in examples given in this Chapter is based on only one training example, where the activations of all of the input layer neurons are set to minimal value, i.e. -1, and the activations of all of the output layer neurons are supposed to be maximal values, i.e. 1, after the stabilisation of the neural network. It is highly probable that this method will not give expected or "good" abductive hypotheses in some cases of abductive problems. Therefore, it is another goal for future works to establish its boundaries and check, if other training examples would yield better results. Finally, there are also other methods of neural networks training, like for example genetics algorithms [38], that can turn out as more efficient and give better or more stable results.

Chapter 6
Abductive reasoning models and implementations

In the first Chapter (p. 1) it was underlined that the concept of abductive reasoning can be constructed in many different ways. Differences in approaches are reflected in the abductive procedures, and finally, in abductive hypotheses that can be obtained by means of those procedures. However, two abductive procedures can produce different abductive hypotheses, despite the fact that there is no difference between them in the general approach to the abduction.

The purpose of this Chapter is to describe a couple of different approaches to abductive reasoning along with different abductive procedures in such a way that it is possible to compare them with the abductive procedure presented in this work. Firstly I will describe and compare Abductive Logic Programming. The reason is that this procedure uses the same formal structures as the approach presented in this work. Afterwards I am going to move to two abductive procedures that are based on the $C\text{-}IL^2P$ system. Finally, I will describe two additional approaches to the abductive reasoning that are based on the classical propositional logic: the first generates one abductive hypothesis that has certain "good" properties, like for example consistency with the knowledge base or minimality, while the second produce huge number of abductive hypotheses in the first step and then screens them out with multiple filters until the hypotheses that left have desired properties.

6.1 Abductive Logic Programming

Abduction in Abductive Logic Programming (ALP) is understood similarly as in the algorithmic approach that is mentioned at the beginning of the first Chapter of this work (see p. 1). Given theory T and sentence G, which is an

observation, we are seeking for the set Δ of sentences (explanation[1] for G) that fulfil the following criteria [28]:

1. $T \cup \Delta \vDash G$.
2. $T \cup \Delta$ satisfies IC.

IC stands for integrity constraints and is a set of conditions that we would like our abductive explanation to meet. The second condition generalises the idea that the abductive hypothesis should be consistent with the knowledge base.

The Abductive Logic Programming will be understood here as an extension of logic programming [28]. As there were several semantics and proof procedures proposed, we will use the approach given by Kakas and Mancarella [29], similarly as in the work of Kakas, Kowalski and Toni [28], which is the base for this section.

The approach described in [29] defines the semantics for Abductive Logic Programming by extending the stable model semantics for logic programs [20]. Therefore, we are going to describe the stable models first.

Definition 6.1. Let $\mathcal{P}$ be a general logic program and $I_{\mathcal{P}}$ a Herbrand interpretation for $\mathcal{P}$, i.e. $I_{\mathcal{P}} \subseteq B_{\mathcal{P}}$. $\mathcal{P}_{I_{\mathcal{P}}}$ is a program obtained from $\mathcal{P}$ in the following way:

1. Delete each Horn clause from $\mathcal{P}$ that has a negative literal $\sim a_i$ in its body, while $a_i \in I_{\mathcal{P}}$.
2. Delete all negative literals in the bodies of the remaining Horn clauses.

A logic program $\mathcal{P}_{I_{\mathcal{P}}}$ is a definite logic program, i.e. it does not contain negations, therefore, it has the least Herbrand model $M_{\mathcal{P}_{I_{\mathcal{P}}}}$. If it is the case that the least Herbrand model $M_{\mathcal{P}_{I_{\mathcal{P}}}}$ coincides with $I_{\mathcal{P}}$, then I is a *stable set* of $\mathcal{P}$, and every stable set of a logic program is a minimal Herbrand model for this logic program [20].

Definition 6.2. Let $\mathcal{P}$ be a logic program and $I_{\mathcal{P}}$ a Herbrand interpretation for $\mathcal{P}$. $I_{\mathcal{P}}$ is a *stable model* for $\mathcal{P}$ iff $I_{\mathcal{P}} = M_{\mathcal{P}_{I_{\mathcal{P}}}}$.

Now we can move to the Abductive Logic Programming. In general, there are three ingredients that together create an abductive framework $\langle \mathcal{P}, A, IC \rangle$: the theory of the form of a general logic program $\mathcal{P}$, the set of abducibles A, and the integrity constraints IC, which is a set of rules that prevent from the addition of "unwanted" information to our knowledge base, or they represent desired properties of the logic program. The set of abducibles A contains atoms from the Herbrand base $B_{\mathcal{P}}$ for the program $\mathcal{P}$ and as in [28] we are going to assume that atoms from A do not appear as a head of any Horn clause in $\mathcal{P}$.

Now we are going to describe the notion of a generalised stable model of the abductive framework $\langle \mathcal{P}, A, IC \rangle$ [28].

[1] The term *explanation* is used by authors in [28] as a replacement for *abductive hypothesis*.

Definition 6.3. Let $\langle \mathcal{P}, A, IC \rangle$ be an abductive framework and $\Delta \subseteq A$ be a set of atoms. $M(\Delta)$ is a *generalised stable model* of $\langle \mathcal{P}, A, IC \rangle$ iff:

- $M(\Delta)$ is a stable model of $\mathcal{P} \cup \Delta$, and
- $M(\Delta) \models IC$.

The set Δ from the above definition is an *abductive extension* of the logic program $\mathcal{P}$ and given a query Q we can call Δ an *abductive explanation* of Q, if $M(\Delta)$ is a generalised stable model of $\langle \mathcal{P}, A, IC \rangle$ and the query Q is a logical consequence of $M(\Delta)$.

Definition 6.4. Let $\langle \mathcal{P}, A, IC \rangle$ be an abductive framework, Q a query, and $\Delta \subseteq A$ a set of atoms. Δ is an *abductive explanation* of Q iff:

- $M(\Delta)$ is a generalised stable model of $\langle \mathcal{P}, A, IC \rangle$, and
- $M(\Delta) \models Q$.

Let us consider the following example that is taken from [28, pp. 34–35]. The abductive framework $\langle \mathcal{P}, A, IC \rangle$ is the following:

$$\mathcal{P} = \{p \leftarrow a \ ; \ q \leftarrow b\} \qquad A = \{a, b\} \qquad IC = \{p \leftarrow q\} \tag{6.1}$$

and the atom we want to explain is p. We have the following three possible abductive explanations (subsets of the set of abducibles A): $\Delta_1 = \{a\}$, $\Delta_2 = \{b\}$ and $\Delta_3 = \{a, b\}$. Both sets, Δ_1 and Δ_3, are in fact abductive explanations of p. The reason is that for the interpretation $M(\Delta_1) = \{a, p\}$, which is a stable model of $\mathcal{P} \cup \Delta_1$, the set of integrity constraints IC is satisfied, therefore, $M(\Delta_1)$ is a generalised stable model of the abductive framework $\langle \mathcal{P}, A, IC \rangle$, and further more, p is *true* for the interpretation $M(\Delta_1)$, i.e. $M(\Delta_1) \models p$. Similar situation occurs when we take the set Δ_3 under consideration. The set of integrity constraints IC is satisfied for the interpretation $M(\Delta_3) = \{a, b, p, q\}$, which is a stable model for $\mathcal{P} \cup \Delta_3$, therefore, $M(\Delta_3)$ is a generalised stable model of the abductive framework $\langle \mathcal{P}, A, IC \rangle$, and in the same time $M(\Delta_3) \models p$. Let us now analyse the last remaining set $\Delta_2 = \{b\}$. The stable model for $\mathcal{P} \cup \Delta_2$ is the interpretation $M(\Delta_2) = \{b, q\}$, but for $M(\Delta_2)$ the set of integrity constraints is not satisfied, therefore $M(\Delta_2)$ is not a generalised stable model for the abductive framework $\langle \mathcal{P}, A, IC \rangle$.

In order to find abductive hypotheses the abductive proof procedure was developed [28, subsec. 5.2]. The procedure consists of two phases: an abductive phase (i.e. the standard SLD-resolution) that generates and collects abductive hypotheses and a consistency phase that checks those hypotheses w.r.t. the integrity constraints and succeeds if all the branches of the search space fail finitely. The proof procedure for ALP is an extension of the abductive proof procedure for negation as finite failure (NAF) [9]. Therefore, the transformation of a logic program has to be introduced.

Firstly we create the abductive framework $\langle \mathcal{P}^*, A^*, IC^* \rangle$ for the logic program $\mathcal{P}$ in the following way [28, pp. 20–21]:

- A^* consists of literals p^* that are complementary literals for every p from $\mathcal{P}$,
- $\mathcal{P}^*$ is $\mathcal{P}$ with all negative literals of the form $\sim p$ replaced by p^*,
- IC^* is the set of all integrity constraints of the form: $\neg(p \wedge p^*)$ and $p \vee p^*$.

The abductive framework that we are going to work on is the following: $\langle \mathcal{P}^*, A \cup A^*, IC \cup IC^* \rangle$. The set of abducibles Δ is a subset of $A \cup A^*$ that satisfies integrity constraints $IC \cup IC^*$.

I will use similar example as in [28, pp. 37–38] to demonstrate the way it works:

$$\mathcal{P} = \{p \leftarrow \sim q \ ; \ q \leftarrow \sim r\} \qquad A = \{q, r\} \qquad IC = \emptyset \tag{6.2}$$

After the transformation described above we obtain the following abductive framework $\langle \mathcal{P}^*, A \cup A^*, IC \cup IC^* \rangle$, where:

$$\mathcal{P}^* = \{p \leftarrow q^* \ ; \ q \leftarrow r^*\} \quad A^* = \{p^*, q^*, r^*\} \quad IC^* = \{ \begin{array}{r} \neg(p \wedge p^*), p \vee p^*, \\ \neg(q \wedge q^*), q \vee q^*, \\ \neg(r \wedge r^*), r \vee r^* \} \end{array}$$

The procedure is shown in the Figure 6.1. We start with the query $\leftarrow p$ and perform the abductive phase, which is the standard SLD-resolution. We move to the atom q^*, which is the "cause" for the atom p, and we assume that q^* occurs, i.e. we add to the set Δ abducible q^*. Now we have to perform the integrity check for the last added abducible, i.e. q^*, thus we move along the snake arrow to change the phase. The resolution of q^* with the integrity constraint $\neg q \vee \neg q^*$ results in getting $\neg q$, which is equivalent to $\leftarrow q$. In order to show that the integrity constraints are not violated we have to reason backwards in SLD-fashion and show that all of the branches end in failure (failure is depicted by a black square in the Figure 6.1). The failure of q is possible only when r^* fails, which further means that r succeeds, because of the integrity constraint $r^* \vee r$. This pushes us again to the abductive phase, along the dotted line, in order to prove r. The atom r is not a head of any Horn clause from program $\mathcal{P}$, therefore, we can add r to the set Δ as an abducible and check if it violates any integrity constraints, thus we change the phase into the integrity check once more by moving along the second snake arrow. There are no domain specific integrity constraints, since $IC = \emptyset$. In such cases, when the abducible is a positive literal, the integrity constraint just checks if the complementary literal—in our situation r^*—does not belong to the set of abducibles Δ. If it is the case that the complementary literal does not belong to the set of abducibles Δ, then the current abductive phase ends with failure and the abducible can be added to the set Δ, which in our situation is r. Since r is in our abductive set, then we can prove r, hence the abductive phase ends with success (success is depicted by an empty square in the Figure 6.1). By proving r we proved also that r^* fails finitely, therefore, the integrity check phase that was caused by the abducible

q^* ends with failure. Finally, by showing in the integrity check phase that q fails finitely we proved that the abducible q^* can be assumed and the whole procedure ends with success and the set of abducibles $\Delta = \{q^*, r\}$, which is an explanation for the atom p.

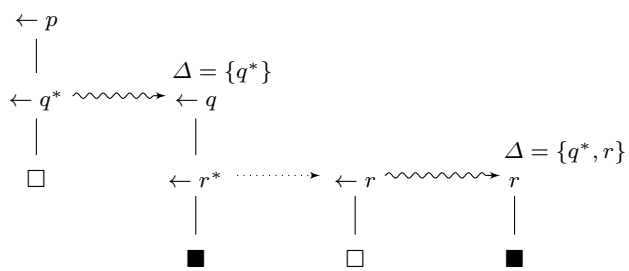

Fig. 6.1 The abductive procedure for the example 6.2.

6.1.1 Comparison

The first difference between ALP and the abductive procedure described in this work lays in the approach to the abductive reasoning and hypotheses that can be obtained. The ALP procedure allows only for the addition of information to the knowledge base, while the abductive procedure presented in this work creates possibility for the addition and removal of information from the given knowledge base. Additionally, the set of abducibles in ALP procedure has to be given by the "user", must contain only atoms and cannot be different from the information already available in the knowledge base, i.e. the set of abducibles is the subset of atoms that are present in the logic program. The same restrictions concern the abductive problem, which must be of the form of a set of atoms that are already available in the logic program. On the contrary, in case of the abductive procedure given in this work there is no predefined set of abducibles, the abductive goal can be of the form of atoms and Horn clauses, and it is allowed for it to contain information that is not available in the knowledge base, i.e. the abductive goal possibly contains atoms that do not belong to the Herbrand base of the logic program that serves as the initial knowledge base. Therefore, ALP does not allow to form new knowledge on the contrary to the abductive procedure given in this work. This is a result of the approach to a logic programs in both procedures—ALP uses logic program and abductive proof procedure defined for it as a source of abductive hypotheses, while in approach presented in this work a logic program is treated only as a "container" for information and therefore, it can be changed during the whole process.

We can apply ALP procedure to the examples from the Chapter 5 and see that in cases where the abductive goal is in the form of an atom, while the whole logic program does not contain negation, obtained abductive hypotheses are similar or exactly the same. Let us consider the following example (see p. 107):

$$\begin{aligned}\mathcal{P}_1 = \{ \quad & a_1 \leftarrow a_2, a_3 \\ & a_2 \leftarrow a_3 \\ & a_2 \leftarrow a_1 \quad \}\end{aligned}$$

where the atom that is an abductive goal is a_1 (in the mentioned example 5.1 the abductive goal is in the form of a fact $a_1 \leftarrow$, but this makes no difference). Then both abductive procedures return the same result, i.e. an abductive hypothesis a_3. What should be noted here is that ALP procedure could produce a hypothesis that consists of two atoms, namely a_2 and a_3, but the restriction imposed on the set of abducibles prevents from a situation where one abducible can be obtained from another one.

Similar situation occurs when the example 5.2 is concerned. In this case the logic program looks as follows:

$$\begin{aligned}\mathcal{P}_2 = \{ \quad & a_1 \leftarrow a_2 \\ & a_1 \leftarrow a_3 \\ & a_4 \leftarrow a_5 \\ & a_3 \leftarrow a_5 \\ & a_2 \leftarrow a_3 \quad \}\end{aligned}$$

and the abductive goal is the same, i.e. an atom a_1. A "good" abductive hypothesis (as it is described in the Section 4.3 on p. 98) is the atom a_5, which is exactly the result of ALP procedure. This is also the most common abductive hypothesis obtained by means of the abductive procedure described in this work (see the Section 5.2 on p. 107). Remark from the previous example holds as well in this case, i.e. the only reason the ALP procedure does not produce an abductive hypothesis a_2 is the mentioned restriction on the set of abducibles.

There is also a difference between ALP and the abductive procedure described in this work that concerns the form of abductive hypotheses. ALP procedure produces abductive hypotheses in the form of sets of atoms. Such hypothesis cannot be added to a logic program, since by definition a logic program is a set of Horn clauses (we would have to change them into facts). Therefore, authors of ALP call those sets of atoms explanations for abductive goal, what is understood as follows: if an explanation occurs long with the clauses from a logic program, then the abductive goal occurs. On the contrary, the abductive procedure described on this work produces abductive

hypotheses of the form of Horn clauses (or their removal) and can be directly added to the logic program that serves as a knowledge base.

Finally, ALP procedure cannot handle abductive problems that are of the form of a Horn clause. As it is shown in the Section 5.2 the abductive procedure described in this work is able to produce abductive hypotheses, where the knowledge base is of the form of the logic program $\mathcal{P}_2$ and the abductive goal is the clause $a_4 \leftarrow a_2$.

6.2 Abduction in $C\text{-}IL^2P$

There are two solutions based on the $C\text{-}IL^2P$ system that generate abductive hypotheses. Both are described in [15]: the first one uses Connectionist Modal Logic (CML) and the second one implements the Abductive Logic Programming approach. I am going to describe both procedures in the following two subsections and at the end I am going to compare them with the abductive procedure introduced in this work.

6.2.1 Using Connectionist Modal Logic (CML)

Neural networks that are created from a logic programs by means of the translation algorithm [14] described in Subsection 4.1.1.1 can perform massively parallel deduction from atoms in the bodies of the Horn clauses in a given logic program into their heads. We could reverse this process, thus we would check what are the "causes" for our atoms that are heads of the Horn clauses from the given logic program. The main assumption here is "that abduction is a form of reverse deduction..." [15, pp. 12–13].

Unfortunately, we cannot just run the neural network in reverse and expect that we obtain all possible inputs for a given output. Therefore, this approach is based on the Connectionist Modal Logic (CML) [18] that enables to translate logic programs with modalities (i.e. structures that represent *necessity* as $\Box$ and *possibility* by $\Diamond$) [39, 51] into neural networks.

For a given logic program $\mathcal{P}$ the abductive procedure is the following [15, pp. 18–19]:

1. Create a world w_i for each atom from the set $B^h_{\mathcal{P}}$, i.e. the set of all heads of the Horn clauses from $\mathcal{P}$.
2. For each Horn clause h_j from $\mathcal{P}$ create a modal rule:

$$w_i : a_i \rightarrow \Diamond(a_k, \ldots, a_l)$$

3. Create accessibility relation $R(w_j, w_i)$ whenever a_i is in the body of a clause in $\mathcal{P}$ whose head is a_j.

4. Build a neural network by means of the translation algorithm for CML [18].
5. For each unconnected output neuron of the form $\Diamond(a_k, \ldots, a_l)$ in the ensemble[2], create a new network with output neurons $a_k, \ldots, a_l$ so that neurons $a_k, \ldots, a_l$ are activated if neuron $\Diamond(a_k, \ldots, a_l)$ is activated.
6. Connect each output neuron a_i in the ensemble to any corresponding output neuron with the same name, i.e. a_i, in any other network of the ensemble such that one neuron is activated whenever the other is activated.

As a result we obtain a neural network that is an ensemble of smaller neural networks that represent the situation occurring in each possible world. The idea is visible in the example given below, were the structure of the neural network is shown in the Figure 6.2.

Let us consider the following example taken from [15, pp. 18–19], where the logic program is the following:

$$\begin{aligned} \mathcal{P} = \{ \quad & x \leftarrow a, b \\ & x \leftarrow c \\ & y \leftarrow x \quad \} \end{aligned} \tag{6.3}$$

that is translated into the following modal logic program (steps 1 and 2):

$$\begin{aligned} \mathcal{P}^M = \{ \quad & w_1 : x \rightarrow \Diamond(a, b) \\ & w_1 : x \rightarrow \Diamond c \\ & w_2 : y \rightarrow \Diamond x \quad \} \end{aligned}$$

along with the accessibility relation $R(w_2, w_1)$. In order to resolve $\Diamond(a, b)$ and $\Diamond c$ from the world w_1 we have to add the following two possible worlds: w_3, where we have a and b, and w_4, where we have c, and the relations $R(w_2, w_3)$ and $R(w_2, w_4)$ (step 3).

Now we can create a neural network for the modal logic program $\mathcal{P}^M$ and additional accessibility relations (steps 4, 5 and 6)—the final result is shown in the Figure 6.2.

The neural network is created in such a way that the activation of the neuron associated with y results in the activation of neurons in the output layer that are associated with $\Diamond(a, b)$ and $\Diamond c$, and furthermore, neurons that are associated with a, b and c.

[2] Each possible world is represented as a three-layered feedforward neural network and the accessibility relation is represented by connections that run from one world to another. In such case the overall structure is called an ensemble of three-layered neural networks rather than a single multi-layered neural network.

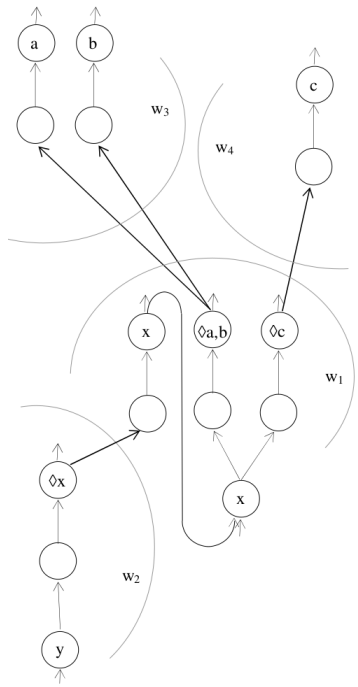

Fig. 6.2 Neural network for the modal logic program $\mathcal{P}^M$ [15, p. 19].

6.2.2 Using Abductive Logic Programming

This approach is based on the propositional version of Abductive Logic Programming [28], which was described in the Section 6.1. The abductive procedure proposed by Garcez et al. [15] applies the four ingredients of ALP, i.e. the theory T, which is a logic program, the goal G, which is a set of atoms, the integrity constraints IC, which is a set of clauses that determines combinations of atoms that if all being *true* lead to the contradiction, and the set of abducibles A, along with three additional structures that are shown in the Figure 6.3.

Elements of the Figure 6.3:

- T—a logic program that is translated into a neural network by means of the translation algorithm described in the Subsection 4.1.1.1,
- IC—representation of the integrity constraints,
- G—representation of the goal atom,
- $LOGIC$—the module that decides if the goal is reached,
- $CLOCK$—the module that waits till the neural network in T stabilise itself,

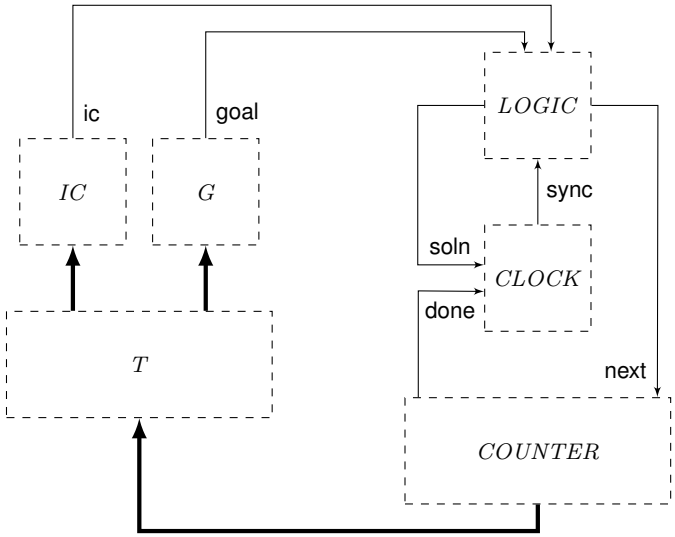

Fig. 6.3 The scheme of the abductive procedure in the *C-IL*^2P system modelling the Abductive Logic Programming approach [15, p. 24].

- *COUNTER*—the module that produces all possible combinations of the *true* and *false* atoms from the set of abducibles A.

The neural network in T that represents our logic program is connected with the module G, that represents the goal atom, and the module IC, that represents integrity constraints, in the following way: if IC (G, respectively) contains atoms that are associated with neurons neurons from the output layer of T, then those output layer neurons are connected with IC (G, respectively). The modules IC and G send signals ic and goal to the module $LOGIC$ if there are some integrity constraints violated or the goal is reached, respectively. Now, if the module $LOGIC$ receives signal sync, then it sends signal soln if on the input it has active signal goal and not active signal ic, otherwise, it sends signal next. The first case describes the situation, where the goal is reached and the integrity constraints are satisfied, i.e. the output layer of the neural network T does not change any more, hence the signal sync from the module $CLOCK$ is active, and neurons from the output layer of T that are associated with atoms from the goal G are active, while neurons from the output layer that are associated with atoms from IC are not active (or are active in a combination that is allowed by integrity constraints). The module $CLOCK$ sends signal sync when the time that is needed for the neural network T passed by and signals soln and done are inactive, i.e. there is no solution yet and there are still combinations of the abducibles to check. We can move to the module $COUNTER$ which generates the binary output, where each binary digit represents an abductible, every time it receives signal next. The output signals of the form of the binary digits are fed to the neural

network T to the input layer neurons that are associated with abducibles. If all combinations of abductibles were checked signal done for the *CLOCK* module is activated.

It is easy to see that the whole system stops in two cases: the goal is reached while the integrity constraints are satisfied, or there are no more combinations of abducibles to check. In the first case the neural network T is stable and the activation of the input layer neurons is the obtained abductive hypothesis. The second case describes the situation where the abductive hypothesis cannot be generated by means of the provided set of abducibles and integrity constraints.

All of the modules can be implemented as a neural network that is a result of the translation of the sets of Horn clauses, as it is described in [15, pp. 25–28] and with more details in [43].

Let us consider the following example taken from [15, p. 23]:

$$\begin{aligned} T &= \left\{ \begin{array}{ll} battery\_flat & \rightarrow wont\_start \\ fuel\_empty & \rightarrow wont\_start \\ wet\_day & \rightarrow battery\_flat \\ fan\_broke & \rightarrow overheat \\ lights\_on & \end{array} \right\} \\ G &= \{wont\_start\} \\ IC &= \{battery\_flat, lights\_on \rightarrow \bot\} \\ A &= \{fan\_broke, fuel\_empty, wet\_day\} \end{aligned} \tag{6.4}$$

Assuming that the abducibles are arranged in the order as in the A set from example 6.4, i.e. the 001 *COUNTER* output state corresponds to the abducible *wet_day* be the only *true*, 010 corresponds to the abducible *fuel_empty* be the only *true* and 100 corresponds to the abducible *fan_broke* be the only *true*, then the system stabilise at the *fuel_empty* solution, which is the abductive hypothesis.

6.2.3 Comparison

The abductive procedure that exploits the method of the Connectionist Modal Logic is described as a reverse deduction that is conveyed from atoms that are heads of Horn clauses from a given logic program to atoms that form the bodies of those Horn clauses. There is no method of "picking the right ones" atoms that are considered *true* after the network stabilisation, therefore, when we consider the example 5.1:

$$\mathcal{P}_1 = \{ \quad a_1 \leftarrow a_2, a_3$$
$$a_2 \leftarrow a_3$$
$$a_2 \leftarrow a_1 \quad \}$$

where the abductive goal is atom a_1, all of the atoms a_1, a_2 and a_3 are considered to be *true*. In such situation it is hard to compare obtained results. We could also try to run the abductive procedure described in this work for the example 6.3. If we assume that the abductive goal is of the form of atom y, then the procedure would most likely produce a hypothesis of the form of the fact $c \leftarrow$ or two facts $a \leftarrow$ and $b \leftarrow$, depending on the factors that were chosen for the translation algorithm form neural networks to logic programs, as it was in the case of examples from the Section 5.2 (see p. 107).

The limitations of this method concern the fact that the obtained abductive hypotheses and the abductive goal are of the form of atoms that are already present in the logic program.

In case of the abductive procedure that is an implementation of the Abductive Logic Programming approach the differences are the same as it was described in the Subsection 6.1.1. Additionally, the neural structure that is constructed in order to compute abductive hypotheses is much smaller in the case of the procedure described in this book, than the procedure that implements Abductive Logic Programming. However, work of Garcez et al. show that it is possible to implement the set of integrity constraints in the neural network using the translation algorithm that was described in the Section 4.1.2. This open possibilities for the abductive procedure described in this book in the future work, where the knowledge base is translated into a neural network alongside with the set of integrity constraints.

6.3 Abduction in Abductive Question Answer System

Abductive Question Answer System (AQAS) [6] implements abductive reasoning as it is understood in the algorithmic approach (see Introduction on p. 1), i.e. the abductive hypothesis fills the deductive gap between the knowledge base Γ and the abductive goal Δ. The idea here is that the initial, potentially complex, abductive problem can be decomposed into simpler problems, where the combined solutions for the simpler problems forms the solution for the initial problem.

The system is formalised by means of the Inferential Erotetic Logic for Classical Propositional Logic, where the Socratic Proofs [34, 62] is the proof method. The language used is $\mathcal{L}^{?}_{\vdash CPL}$, which is an extension of the language of the CPL by the following symbols: $\vdash$ (turnstile, intuitively it stands for the derivability relation in CPL), question mark, comma and semicolon. The basic structure in the system is a question formalised as a finite and non-empty sequence of sequents of $\mathcal{L}^{?}_{\vdash CPL}$ that consist of finite and non-empty

sequences of formulas of the CPL language. If a sequent consists only of literals, then it is called *atomic*, and if a question consists only of atomic sequents, then it is called a *minimal question*.

Definition 6.5. Let Γ and Δ be finite and non-empty sequences of formulas of $\mathcal{L}_{CPL}$. An *atomic declarative formula* of $\mathcal{L}^{?}_{\vdash CPL}$ or *sequent* is the following structure:

$$\Gamma \vdash \Delta$$

Definition 6.6. Let Φ be a finite and non-empty sequence of sequents. Then *question* or *erotetic formula* of $\mathcal{L}^{?}_{\vdash CPL}$ is of the following form:

$$?(\Phi)$$

Intuitively, this represents a question concerning the CPL-derivability of the information given in the consequents of sequents in Φ from antecedents of those sequents. For example, a question $?(S \vdash A)$ asks about CPL-derivability of A from S [62, p. 23].

As in the case of the logic programs, the semicolon is used to separate sequents in questions, which is visible in the examples given in the later part of this Section.

There are two kinds of sequents: *closed* and *open*. A sequent of the form $\Gamma \vdash \Delta$ is closed iff one of the following conditions is met:

1. Γ or Δ contains F and $\neg F$ (complementary formulas).
2. Γ and Δ contain F.

where F is a formula of the CPL. If sequent is not closed, then its open. The intuition here is the following, if a sequent is closed, then, if all formulas from Γ are *true*, then at least one formula in Δ is *true*.

An abductive problem (*abductive question*) occurs when the sequent in the initial question is open. This is the starting point for the whole procedure. Now we want to transform the (potentially) complicated problem into simpler problems. This is obtained by means of $\mathbb{E}^{CPL}$, which is an erotetic calculus for CPL. The rules of $\mathbb{E}^{CPL}$ are shown in Tables 6.1 and 6.2, where the α, β-notation was used [10]. At the end of a transformation of a question a minimal question is obtained, which is a sequence of atomic sequents, i.e. sequents which contain only literals.

Table 6.1 α/β—formulas [6].

α	α_1	α_2	β	β_1	β_2
$A \wedge B$	A	B	$\neg(A \wedge B)$	$\neg A$	$\neg B$
$\neg(A \vee B)$	$\neg A$	$\neg B$	$A \vee B$	A	B
$\neg(A \rightarrow B)$	A	$\neg B$	$A \rightarrow B$	$\neg A$	B

Table 6.2 Rules of $\mathbb{E}^{CPL}$ [6].

$$\frac{?(\Phi;\Gamma,\alpha,\Gamma'\vdash\Delta;\Psi)}{?(\Phi;\Gamma,\alpha_1,\alpha_2,\Gamma'\vdash\Delta;\Psi)}\ \mathbf{L}_\alpha \qquad \frac{?(\Phi;\Gamma\vdash\Delta,\alpha,\Delta';\Psi)}{?(\Phi;\Gamma\vdash\Delta,\alpha_1,\Delta';\Gamma\vdash\Delta,\alpha_2,\Delta';\Psi)}\ \mathbf{R}_\alpha$$

$$\frac{?(\Phi;\Gamma,\beta,\Gamma'\vdash\Delta;\Psi)}{?(\Phi;\Gamma,\beta_1,\Gamma'\vdash\Delta;\Gamma,\beta_2,\Gamma'\vdash\Delta;\Psi)}\ \mathbf{L}_\beta \qquad \frac{?(\Phi;\Gamma\vdash\Delta,\beta,\Delta';\Psi)}{?(\Phi;\Gamma\vdash\Delta,\beta_1,\beta_2,\Delta';\Psi)}\ \mathbf{R}_\beta$$

$$\frac{?(\Phi;\Gamma,\neg\neg A,\Gamma'\vdash\Delta;\Psi)}{?(\Phi;\Gamma,A,\Gamma'\vdash\Delta;\Psi)}\ \mathbf{L}_{\neg\neg} \qquad \frac{?(\Phi;\Gamma\vdash\Delta,\neg\neg A,\Delta';\Psi)}{?(\Phi;\Gamma\vdash\Delta,A,\Delta';\Psi)}\ \mathbf{R}_{\neg\neg}$$

Definition 6.7. A finite sequence of questions $\mathbf{s}=\langle s_1,\ldots,s_n\rangle$ is a *Socratic transformation* (*s-transformation*) of a question $?(\Phi)$ by means of $\mathbb{E}^{CPL}$ iff the following conditions hold:

1. $s_1 = ?(\Phi)$.
2. s_i $(1 < i \leq n)$ results from s_{i-1} by an application of a rule of $\mathbb{E}^{CPL}$.

An s-transformation which has as the last question a minimal question is called *complete*. It is clear that if the initial question Q is an abductive question, i.e. a sequent in question Q is open, then the last question of the complete s-transformation of question Q contains at least one open sequent, which is atomic. The rules of $\mathbb{E}^{CPL}$ are invertible, therefore, if we add some formulas to all of the sequents that are open at the end of the s-transformation in such a way that they become closed, then the addition of all of those formulas to the initial question closes it as well. Combining those two facts together, i.e. the simple form of the last sequents of a complete s-transformation of an abductive question and the property that grants us that the closure of the open sequents at the end of the s-transformation leads to the closure of the initial question, we have obtained the way to resolve a given abductive problem by decomposing it into simpler problems that are easier to resolve.

In order to close an atomic sequent we are going to use *abductive rules* shown in the Table 6.3. There are two ways we can follow: either we are going to make the antecedent of the sequent contradictory, or we put a connection between a literal from the antecedent of the sequent and a literal from the consequent of the sequent. The overlined literal $\bar{l}$ represents the complementary literal for l.

Table 6.3 Abductive rules. [6]

$$\frac{?(\Phi\ ;\ \Theta_1,l,\Theta_2\vdash\Theta_3\ ;\ \Psi)}{\bar{l}}\ \mathbf{R}^1_{abd} \qquad \frac{?(\Phi\ ;\ \Theta_1,l,\Theta_2\vdash\Theta_3,k,\Theta_4\ ;\ \Psi)}{l\rightarrow k}\ \mathbf{R}^2_{abd}$$

At this stage, applying abductive rules to random literals could result in an abductive hypothesis that makes initial knowledge base Γ contradictory or is

too strong, i.e. the abductive goal is obtainable from the abductive hypothesis alone. In order to prevent situations of these kinds we proposed restrictions for both abductive rules. The base for those restrictions are *downward saturated sets* (or *Hintikka sets*) and *dual downward saturated sets* (or *dual Hintikka sets*).

Definition 6.8. Let Γ be a sequence of formulas of CPL. By a *downward saturated set* (*Hintikka set*) that corresponds to the Γ we mean a set $\mathfrak{U}_\Gamma$ that fulfils the following conditions:

1. If $F \in \Gamma$, then $F \in \mathfrak{U}_\Gamma$.
2. If $\alpha \in \mathfrak{U}_\Gamma$, then $\alpha_1 \in \mathfrak{U}_\Gamma$ and $\alpha_2 \in \mathfrak{U}_\Gamma$.
3. If $\beta \in \mathfrak{U}_\Gamma$, then $\beta_1 \in \mathfrak{U}_\Gamma$ or $\beta_2 \in \mathfrak{U}_\Gamma$.
4. If $\neg\neg F \in \mathfrak{U}_\Gamma$, then $F \in \mathfrak{U}_\Gamma$.
5. Nothing more belongs to $\mathfrak{U}_\Gamma$ except for those formulas that enter $\mathfrak{U}_\Gamma$ on the grounds of conditions 1–4.

Definition 6.9. Let Δ be a sequence of formulas of CPL. By a *dual downward saturated set* (*dual Hintikka set*) that corresponds to the Δ we mean a set $\mathfrak{W}_\Delta$ that fulfils the following conditions:

1. If $F \in \Delta$, then $F \in \mathfrak{W}_\Delta$.
2. If $\alpha \in \mathfrak{W}_\Delta$, then $\alpha_1 \in \mathfrak{W}_\Delta$ or $\alpha_2 \in \mathfrak{W}_\Delta$.
3. If $\beta \in \mathfrak{W}_\Delta$, then $\beta_1 \in \mathfrak{W}_\Delta$ and $\beta_2 \in \mathfrak{W}_\Delta$.
4. If $\neg\neg F \in \mathfrak{W}_\Delta$, then $F \in \mathfrak{W}_\Delta$.
5. Nothing more belongs to $\mathfrak{W}_\Delta$ except for those formulas that enter $\mathfrak{W}_\Delta$ on the grounds of conditions 1–4.

We are interested in all of those downward saturated sets that are consistent, i.e. there exists a Boolean valuation for which all formulas in such set are *true*. Taking into account the definition of the downward saturated set we can say that such set is consistent iff it does not contain complementary literals. The collection of all consistent downward saturated sets can be understood as a collection of all those valuations that make all of the formulas that belong to the knowledge base Γ *true*. If we add an abducive hypothesis to every downward saturated set in our collection, then we have an answer if there exists a valuation under which our hypothesis is consistent with the knowledge base.

Definition 6.10. By a *consistency property* corresponding to a sequence of formulas Γ we understand a finite set $\mathfrak{U}^c_\Gamma = \{\mathfrak{U}^1_\Gamma, \ldots, \mathfrak{U}^n_\Gamma\}$, which contains all downward saturated sets for Γ that do not contain complementary literals.

Similarly for the dual downward saturated sets—a dual Hintikka set is dually satisfied if there is a Boolean valuation for which at least one formula in such set is *true*. Therefore, a dual downward saturated set is dually valid iff there is no Boolean valuation for which every formula in such set is *false*. Taking into account the definition of the dual downward saturated set we can

identify dually valid sets as those that contain complementary literals. In such case we are interested in collecting all of those dual downward saturated sets that are not dually valid. Such collection can be understood as a collection of all those valuations that can make the antecedent of the sequent, i.e. the sequence Δ[3], *false*. The reason is that we do not want to create abductive hypotheses that make the consequent of the sequent (Δ) unfalsifiable on the ground of the abductive hypothesis alone, i.e. they do not make all of the dual downward saturated sets in non-validity property dually valid.

Definition 6.11. By a *non-validity property* corresponding to a sequence of formulas Δ we understood a finite set $\mathfrak{W}^{nv}_{\Delta} = \{\mathfrak{W}^1_{\Delta}, \ldots, \mathfrak{W}^n_{\Delta}\}$, which contains all dual downward saturated sets for Δ that do not contain complementary literals.

Now we can define restrictions for the abductive rules $\mathbf{R}^1_{abd}$ and $\mathbf{R}^2_{abd}$ from the Table 6.3 that will grant us that the crated hypothesis is consistent with the knowledge base and that it is not too strong, i.e. the abductive goal is not obtainable from the hypothesis alone. The first kind of restrictions is called *consistency restrictions* and the second kind—*significance restrictions*:

- Consistency restriction on $\mathbf{R}^1_{abd}$:
  There exists a set $\mathfrak{U}_{\Gamma} \in \mathfrak{U}^c_{\Gamma}$ such that $l \notin \mathfrak{U}_{\Gamma}$.
- Consistency restriction on $\mathbf{R}^2_{abd}$:
  There exists a set $\mathfrak{U}_{\Gamma} \in \mathfrak{U}^c_{\Gamma}$ such that $l \notin \mathfrak{U}_{\Gamma}$ or $\overline{k} \notin \mathfrak{U}_{\Gamma}$.
- Significance restriction on $\mathbf{R}^1_{abd}$:
  There exists a set $\mathfrak{W}_{\Delta} \in \mathfrak{W}^{nv}_{\Delta}$ such that $\bar{l} \notin \mathfrak{W}_{\Delta}$.
- Significance restriction on $\mathbf{R}^2_{abd}$:
  There exists a set $\mathfrak{W}_{\Delta} \in \mathfrak{W}^{nv}_{\Delta}$ such that $\bar{l} \notin \mathfrak{W}_{\Delta}$ or $k \notin \mathfrak{W}_{\Delta}$.

Having defined all parts of the system we can now describe the abductive procedure:

1. Given a sequent $\Gamma \vdash \Delta$, where Γ stands for the knowledge base and Δ for the abductive goal, create a complete s-transformation of the question $?(\Gamma \vdash \Delta)$.
2. On the basis of Γ create the consistency property $\mathfrak{U}^c_{\Gamma}$ and on the basis of Δ the non-validity property $\mathfrak{W}^{nv}_{\Delta}$.
3. Take the first open sequent from the last question of the s-transformation created in the first step and close it with either abductive rule, $\mathbf{R}^1_{abd}$ or $\mathbf{R}^2_{abd}$, in such a way that the restrictions concerning this rule are not violated.
4. Remove from the consistency property $\mathfrak{U}^c_{\Gamma}$ all downward saturated sets that become inconsistent after the addition of the created hypothesis. Similarly, remove from the non-validity property $\mathfrak{W}^{nv}_{\Delta}$ all those dual downward

[3] Δ stands for the antecedent of a sequent and intuitively is understood as the information that is not obtainable from the knowledge base, i.e. antecedent of a sequent marked by Γ.

saturated sets that become dually valid after the addition of the created hypothesis.

5. If there are no other open sequents, combine all hypotheses with the conjunction—this is your final abductive hypothesis, otherwise, proceed to the next open sequent and repeat steps 3–4.

In [6] we have a proof of the Theorem, which states that creating abductive hypotheses in a way described above grants us that the final abductive hypothesis is consistent with our knowledge base and that the abductive goal is not obtainable from the hypothesis alone.

Let us consider the following example that illustrate how the abductive procedure works. The example is taken form [6]. We start with the following knowledge base Γ and abductive goal Δ:

$$\Gamma = \langle p \to (z \to q), r \wedge s \rangle$$
$$\Delta = \langle r \to q \rangle$$

The complete s-transformation of the question $?(p \to (z \to q), r \wedge s \vdash r \to q)$ looks as follows (first step):

$$\dfrac{\dfrac{\dfrac{\dfrac{?(p \to (z \to q), r \wedge s \vdash r \to q)}{?(r, s, p \to (z \to q) \vdash r \to q)}\,\mathbf{L}_\wedge}{?(r, s, p \to (z \to q) \vdash \neg r, q)}\,\mathbf{R}_\to}{?(\neg p, r, s \vdash \neg r, q\ ;\ z \to q, r, s \vdash \neg r, q)}\,\mathbf{L}_\to}{?(\neg p, r, s \vdash \neg r, q\ ;\ \neg z, r, s \vdash \neg r, q\ ;\ q, r, s \vdash \neg r, q)}\,\mathbf{L}_\to$$

Following the abductive procedure, we are going now to generate downward saturated sets:

$$\mathfrak{U}^1_\Gamma = \{p \to (z \to q), r \wedge s, r, s, \neg p, z \to q, \neg z, q\}$$
$$\mathfrak{U}^2_\Gamma = \{p \to (z \to q), r \wedge s, r, s, \neg p, z \to q, \neg z\}$$
$$\mathfrak{U}^3_\Gamma = \{p \to (z \to q), r \wedge s, r, s, \neg p, z \to q, q\}$$
$$\mathfrak{U}^4_\Gamma = \{p \to (z \to q), r \wedge s, r, s, \neg p\}$$
$$\mathfrak{U}^5_\Gamma = \{p \to (z \to q), r \wedge s, r, s, z \to q, \neg z, q\}$$
$$\mathfrak{U}^6_\Gamma = \{p \to (z \to q), r \wedge s, r, s, z \to q, \neg z\}$$
$$\mathfrak{U}^7_\Gamma = \{p \to (z \to q), r \wedge s, r, s, z \to q, q\}$$

where consistent downward saturated sets form a consistency property:

$$\mathfrak{U}^c_\Gamma = \{\mathfrak{U}^1_\Gamma, \mathfrak{U}^2_\Gamma, \mathfrak{U}^3_\Gamma, \mathfrak{U}^4_\Gamma, \mathfrak{U}^5_\Gamma, \mathfrak{U}^6_\Gamma, \mathfrak{U}^7_\Gamma\}$$

and dual downward saturated sets:

$$\mathfrak{W}^1_\Delta = \{r \to q, \neg r, q\}$$

where non-valid dual saturated sets form a non-validity property:

$$\mathfrak{W}^{nv}_{\Delta} = \{\mathfrak{W}^{1}_{\Delta}\}$$

Looking at the last question in the s-transformation created in the first step

$$?(\neg p, r, s \vdash \neg r, q \ ; \ \neg z, r, s \vdash \neg r, q \ ; \ q, r, s \vdash \neg r, q)$$

we can see that there are two open sequents:

$$\neg p, r, s \vdash \neg r, q$$
$$\neg z, r, s \vdash \neg r, q$$

Let us take the first open sequent $\neg p, r, s \vdash \neg r, q$ and close it by means of the abductive rule $\mathbf{R}^{1}_{abd}$. We can choose literal $\neg p$ from the antecedent of the sequent and create hypothesis p. We do not violate both restrictions, however, our consistency property shrinks to those downward saturated sets that do not contain $\neg p$:

$$\mathfrak{U}^{c}_{\Gamma} = \{\mathfrak{U}^{5}_{\Gamma}, \mathfrak{U}^{6}_{\Gamma}, \mathfrak{U}^{7}_{\Gamma}\}$$

Having the first open sequent closed we can move to the second open sequent, i.e. $\neg z, r, s \vdash \neg r, q$. We can close it by means of the second abductive rule $\mathbf{R}^{2}_{abd}$: let us take the literal $\neg z$ from the antecedent of the sequent and the literal q from the consequent of the sequent, and from a hypothesis $\neg z \rightarrow q$. This hypothesis does not violate both restrictions for the rule $\mathbf{R}^{2}_{abd}$ since the set $\mathfrak{U}^{7}_{\Gamma}$ does not contain the literal $\neg z$, and the set $\mathfrak{W}^{1}_{\Delta}$ does not contain the literal z.

There are no other open sequents in our question, therefore, we can move to the last step of the abductive procedure and form the final abductive hypothesis, which is a conjunction of both hypotheses generated in previous steps:

$$p \wedge (\neg z \rightarrow q)$$

The addition of the abductive hypothesis $p \wedge (\neg z \rightarrow q)$ to the knowledge base Γ allows us to derive the abductive goal Δ, while in the same time the knowledge base remains consistent and the abductive goal cannot be derived from the abductive hypothesis alone.

6.3.1 Comparison

The approach to the abductive reasoning that is implemented in the Abductive Question-Answer System is similar to the one used in the Abductive Logic Programming. The difference here is that in AQAS abductive hypothesis should be consistent with the knowledge base rather than with the set of integrity constraints. Therefore, at this level the difference between the

abductive procedure introduced in this work and AQAS is similar as in the case of ALP, i.e. allowing abductive hypotheses to be a symmetric difference between the initial knowledge base and the one that results from applying the abductive procedure is a broader approach than allowing abductive hypotheses to be only extra knowledge that is added to the knowledge base.

The starting point is similar in both abductive procedures, i.e. AQAS and the one described in this work. The abductive goal is defined as information that cannot be derived from the knowledge base and the set of abducibles is not given from the beginning. In both cases only one abductive hypothesis is created rather than a huge number of potential candidates that is systematically narrowed down by given criteria, and the hypotheses have desired properties, which are the consistency with the knowledge base and the inability of the abductive goal derivation from the abductive hypothesis alone.

Now we would like to compare results obtained from specific examples, but here lays the following problem. The language and semantics of propositional logic programs can be interpreted in two ways in the CPL that is used in AQAS. We could treat every Horn clause as implication and the negation as failure as classical negation. The other way is to create Clark's completion of a logic program, which it meant to translate it into the language of CPL preserving the unique semantics of logic programs (see the Section 2.4.1 on p. 45). We will try to compare the two abductive procedures in both ways here.

Let us consider the example 5.1 (see p. 107):

$$\begin{aligned} \mathcal{P}_1 = \{ \quad & a_1 \leftarrow a_2, a_3 \\ & a_2 \leftarrow a_3 \\ & a_2 \leftarrow a_1 \quad \} \end{aligned}$$

where the abductive goal is the atom a_1.

In the first case, when we treat logic program $\mathcal{P}_1$ as a set of classical implications, the abductive hypothesis obtained by means of AQAS can be the same as the one obtained in the example 5.1, i.e. atom a_3. However, AQAS is an abductive procedure that returns one solution, but not necessarily the best in terms of the complexity—using abductive rules (see the Table 6.3 on p. 132) along with restrictions (see p. 134) guarantees only that obtained hypotheses are consistent with the initial knowledge base and are not to strong, i.e. the abductive goal is not obtainable from the hypothesis alone.

The situation is different if we make Clark's completion for the logic program $\mathcal{P}_1$ and then perform on the obtained set of CPL formulas the abductive procedure from AQAS. The Clark's completion for the program $\mathcal{P}_1$ looks as follows:

$$\begin{aligned} \mathsf{comp}(\mathcal{P}_1) = \{ \quad & a_1 \leftrightarrow a_2 \wedge a_3 \\ & a_2 \leftrightarrow a_3 \vee a_1 \\ & \neg a_3 \quad \} \end{aligned} \tag{6.5}$$

There is only one model for such set of formulas, where all atoms, i.e. a_1, a_2 and a_3 are mapped to *false*, therefore there is only one Hintikka set which contains atoms a_1, a_2, a_3 preceded with the negation. This means that we cannot create any abductive hypothesis with respect to the restrictions given in 6.3, because the abductive goal contradicts with every model for the knowledge base, and thus after addition of any of hypotheses created by means of abductive rules from the Table 6.3 to the knowledge base it would become inconsistent.

The situation is similar in the second example 5.2 (see p. 107). For the logic program:

$$\begin{aligned} \mathcal{P}_2 = \{ \quad & a_1 \leftarrow a_2 \\ & a_1 \leftarrow a_3 \\ & a_4 \leftarrow a_5 \\ & a_3 \leftarrow a_5 \\ & a_2 \leftarrow a_3 \quad \} \end{aligned}$$

and the abductive goal the same as in the previous example, the abductive procedure from AQAS can generate "good" abductive hypothesis for this case, i.e. atom a_5. However, when the Clark's completion of the program $\mathcal{P}_2$ is concerned:

$$\begin{aligned} \mathsf{comp}(\mathcal{P}_2) = \{ \quad & a_1 \leftrightarrow a_2 \vee a_3 \\ & a_4 \leftrightarrow a_5 \\ & a_3 \leftrightarrow a_5 \\ & a_2 \leftrightarrow a_3 \\ & \neg a_5 \quad \} \end{aligned} \tag{6.6}$$

there is only one model for such collection of formulas, where all atoms are, i.e. $a_1, \ldots, a_5$, are mapped to $false$. As in the former example, there is only one Hintikka set, which contains all atoms occurring in formulas from $\mathsf{comp}(\mathcal{P}_2)$, therefore it is not possible to create abductive hypothesis by means of the abductive rules with respect to the restrictions.

Let us now consider again the logic program $\mathcal{P}_2$ but with the abducive goal of the form of the clause $a_4 \leftarrow a_2$. The "good hypothesis" here is the atom a_5 (see the Section 5.2 on p. 107), and it is obtainable by means of the abductive rules used with respect to the restrictions. In the two previous examples creating Clark's completion for the knowledge base caused problems, because the abductive goal was contradictory to all models for such sets of formulas. In this case the abductive goal becomes obtainable from the knowledge base, therefore there is no need to run the abductive procedure.

The cause of the fact that the Clark's completion made such a difference in the process of abductive hypotheses creation by means of the AQAS procedure may lay in the specificity of used examples. In addition, analysed

examples do not explore the potential of other type of abductive hypotheses that AQAS is able to create, namely those of the form of implications. The reason is that the goal here was to show basic differences and similarities between both approaches rather than making comprehensive comparison. Therefore, it would be very interesting to compare situations, where the "good" hypotheses require those more complicated form with the addition or removal of Horn clauses from logic programs in the corresponding cases for the abductive procedure described in this work.

6.4 Abduction with Synthetic Tableaux Method

The abductive procedure from [30] implements the algorithmic account of abduction [12, p. 88], where the basic logic is classical propositional logic, the abductive procedure that produces abductive hypotheses is the Synthetic Tableaux Method [57] for CPL and then there were defined five criteria that were used in multi-criteria analysis to evaluate obtained abductive hypotheses.

The Synthetic Tableaux Method [56, 57] for CPL is a proof method, where a synthetic tableau for a formula F is created, which in turn is a collection of *synthetic inferences* $\mathbf{s}$ of $F/\neg F$. Synthetic inferences are generated on the basis of sets of basic constituents, i.e. literals occurring in F or their negations. The formulas that occur in a given synthetic inference can be only subformulas of the formula F or their negations. The derivability relation is defined by the rules listed in the Table 6.4.

Given a set of clauses Γ which stands for the initial knowledge base and a formula ϕ a synthetic tableau for a derivation of ϕ from Γ is defined as a family of synthetic inferences of $\Gamma \cup \{\phi\}$ or their negations on the basis of consistent sets of literals occurring in $\Gamma \cup \{\phi\}$ or their negations. A synthetic inference $\mathbf{s}$ of a formula ϕ on the basis of the set Γ is called a *success* iff $\mathbf{s}$ contains $\neg F$, where $F \in \Gamma$, or $\mathbf{s}$ contains ϕ. If it is the case that $\mathbf{s}$ contains all formulas from Γ and the negation of the formula ϕ, i.e. $\Gamma \cup \{\phi\} \subseteq \mathbf{s}$, then we call $\mathbf{s}$ a *failure*. We can say that a set Γ entails a formula ϕ iff there exists a synthetic tableau for the derivation of ϕ on the basis of Γ such that each element of the tableau is a *success* [30]. Since the abductive problem occurs when a given formula is not obtainable form a given knowledge base, we are

Table 6.4 Rules for STM [30].

$$\frac{\neg\alpha}{\alpha \to \beta}\ \mathbf{r}^1_{\to} \qquad \frac{\beta}{\alpha \to \beta}\ \mathbf{r}^2_{\to} \qquad \frac{\alpha, \neg\beta}{\neg(\alpha \to \beta)}\ \mathbf{r}^3_{\to} \qquad \frac{\alpha}{\alpha \vee \beta}\ \mathbf{r}^1_{\vee} \qquad \frac{\beta}{\alpha \vee \beta}\ \mathbf{r}^2_{\vee}$$

$$\frac{\neg\alpha, \neg\beta}{\neg(\alpha \vee \beta)}\ \mathbf{r}^3_{\vee} \qquad \frac{\neg\alpha}{\neg(\alpha \wedge \beta)}\ \mathbf{r}^1_{\wedge} \qquad \frac{\neg\beta}{\neg(\alpha \wedge \beta)}\ \mathbf{r}^2_{\wedge} \qquad \frac{\alpha, \beta}{\alpha \wedge \beta}\ \mathbf{r}^3_{\wedge} \qquad \frac{\alpha}{\neg\neg\alpha}\ \mathbf{r}_{\neg\neg}$$

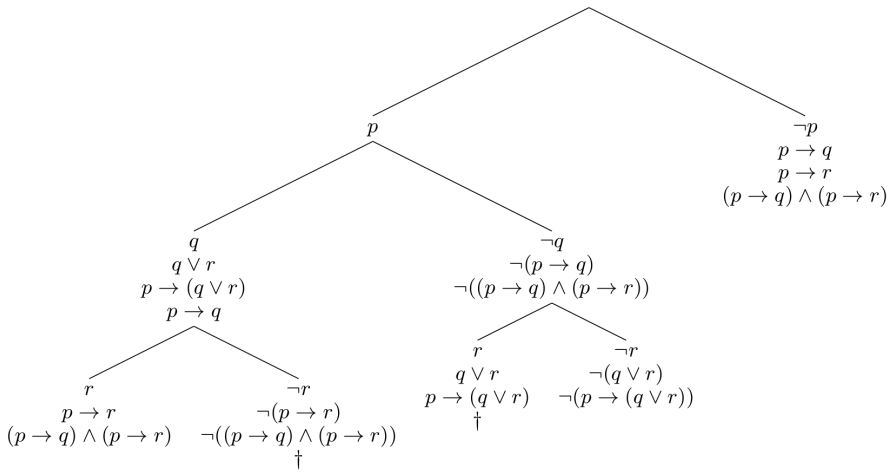

Fig. 6.4 Example of a synthetic tableau [30].

interested only in those synthetic tableau, which contain at least one *failure*. Let us consider the following example taken from [30], where the knowledge base is the following:

$$\Gamma = \{p \to q \vee r\}$$

and the abductive goal is the formula:

$$\phi : (p \to q) \wedge (p \to r)$$

The synthetic tableau for the derivation of ϕ from Γ is shown in the Figure 6.4. A synthetic inference is a sequence of formulas from a given branch, for example:

$$\mathbf{s}_1 = \neg p, p \to q, p \to r, (p \to q) \wedge (p \to r), q, \neg r$$

is a rightmost synthetic inference. Synthetic inferences that are *failures* are marked with the † symbol. In the exemplary synthetic inference given above literals q and r were added, but they are not required to build the synthetic inference $\mathbf{s}_1$, as it can be seen in the Figure 6.4. The situation is different with literal $\neg p$—all formulas in $\mathbf{s}_1$ are derived from this literal and thus it is called an *entangled* literal, which is relevant for $\mathbf{s}_1$ being a *success* or a *failure* [30].

The algorithm that generates the set of abducibles uses the entangled literals "gathered" from *failures*. Given a knowledge base Γ and an abductive goal ϕ, the generation of the set of abducibles takes four steps [30]:

1. A synthetic tableau for a derivation of ϕ on the basis of Γ is created.
2. For each *failure* $\mathbf{s}_i$ (where $i \in [1, \ldots, k]$, and k is the number of *failures* in the synthetic tableau) the entangled literals $L_1^i, \ldots, L_n^i$ of $\mathbf{s}_i$ are extracted.

3. For each failure $\mathbf{s}_i$ a formula $D_i = \neg(L_1^i \wedge \ldots \wedge L_n^i)$ is build. Afterwards, a conjunction $D_1 \wedge \ldots \wedge D_i \wedge \ldots \wedge D_k$ is created.
4. The conjunction $D_1 \wedge \ldots \wedge D_i \wedge \ldots \wedge D_k$ is transformed into the disjunctive normal form $A_1 \vee \ldots \vee A_m$, where each disjunct A_j (or any subset of the set $\{A_1, \ldots, A_m\}$ with elements connected with disjunction) is an abducible for the formula ϕ with respect to the knowledge base Γ.

The set of abducibles obtained by means of the described algorithm is huge and contains unwanted elements, such as e.g. inconsistent sets of literals. However, I will not go here into details of the implementation that allows to "trim" the set of abducibles for a given abductive problem—the reader is sent to the source [30].

Having the set of abducibles we can move to the next step, i.e. the evaluation of abductive hypotheses. There were five criteria [30]:

1. Consistency—if a hypothesis H is consistent with the knowledge base, the hypothesis H gets 1 as evaluation, and 0 otherwise.
2. Significance—if an abductive goal G is entailed by the hypothesis H, the hypothesis H gets 0 as evaluation, and 1 otherwise.
3. Complexity—the number of distinct variables a hypothesis H contains (the smaller the better).
4. Operational complexity—the number of two-argument operators a hypothesis H contains (the smaller the better).
5. Overlapping—the number of variables that occur in both: a hypothesis H and abductive goal G (the smaller the better).

The criteria described above are discussed in more details in the Subsection 4.3.1.

Now we can evaluate each abductive hypothesis and create a classification from the "best" to the "worse" abductive hypothesis, based on the multi-criteria analysis, where we want to maximise the first two criteria, and to minimise the remaining three. For example, for the abductive problem where the knowledge base contains the following formulas:

$$\begin{aligned} p &\rightarrow (q \vee r) \\ \neg t &\rightarrow \neg r \\ q &\rightarrow (s \vee t) \\ s &\rightarrow (\neg t \vee r) \end{aligned}$$

and the abductive goal in the form of atom s, the initial set of abducibles was containing 2^{102400} elements, and after the simplification and evaluation it was narrowed down to just two non-dominated abductive hypotheses:

$$\begin{aligned} p \wedge \neg t \\ q \wedge \neg t \end{aligned}$$

where the simplification process removes from the set of abducibles those hypotheses that are logically equivalent to other abductive hypotheses.

6.4.1 Comparison

The abductive procedure given in [30] is similar to ALP procedure in the approach to the abduction, where the set of abducibles is created at the beginning, and then a method of selection is introduced that allows to pick up only those abducibles that fulfil certain criteria. Therefore, it is different from the abductive procedure proposed in this work (or AQAS), where there is no set of abducibles that is being "trimmed", but only one hypothesis that potentially has desired properties (or in case of AQAS is guaranteed to has those properties). Another point is that the method of evaluation of abductive hypotheses, which employs the multi-criteria analysis based on the five criteria defined in [30] is much more sophisticated than the criteria used in any other abductive procedure described in this work. For that reason it would be very interesting to design such abductive problems that could be analysed by both, the method in [30] and the abductive procedure proposed in this work, and compare obtained abductive hypotheses. Here of course arises the problem of the compatibility of the propositional logic programs and the classical propositional logic, which was described in the Subsection 6.3.1 (see p. 136) devoted for the comparison between AQAS and the abductive procedure proposed in this work.

6.5 Chapter summary

There are many different approaches to the problem of formalisation and implementation of abductive reasoning. The solutions described in this Chapter allow to put the abductive procedure proposed and described in this work in a wider context. The comparisons that were made at the end of each section showed differences and similarities between approaches as far as the obtained abductive hypotheses for particular problems are concerned. In addition, the differences between intuitions and notions concerning abductive reasoning that stand behind each approach were also discussed.

In the first two Sections (p. 119 and 125, respectively) abductive procedures were described that use as formal grounds logic programs. We started with Abductive Logic Programming [28], which is a well known framework for abductive hypotheses creation. The main differences between ALP and the abductive procedure proposed in this work lay in the fact that in the former approach at the beginning of the abducive procedure we are already equipped with the set of abducibles. Additionally, the set of abducibles is

prepared in such a way, that it contains only those abducibles that cannot be derived from other abducibles. The next difference is that the abductive goals and hypotheses are restricted only to atoms that occur in the knowledge base, while by means of the procedure proposed in this work we can consider abductive goals in the form of Horn clauses that do not necessarily contain atoms only from the knowledge base, and create abductive hypotheses of the form of Horn clauses (or their removal).

After ALP procedure we moved to the two approaches to the abductive reasoning that are implemented by means of the $C\text{-}IL^2P$ system [15]. The first one was based on the modal version of logic programs. Since there was no definition of the abductive goal and the abductive hypothesis it was hard to compare this approach with the one proposed one proposed in this work. The second one was the implementation of ALP procedure by means of the $C\text{-}IL^2P$ system, therefore the differences listed in the comparison from the first Section apply also in this case. However, this implementation of ALP procedure shows that it is possible to implement the set of integrity constraints by means of $C\text{-}IL^2P$ system, what can be developed further in the future work.

The third Section was devoted for the description of the Abductive Question-Answer System [6]. It appears that AQAS is very similar to the abductive procedure proposed in this work because of the two following reasons: firstly, both abductive procedures create "good" abductive hypothesis, which may turn out to be the best; secondly, both system are flexible as far as the form of abductive hypotheses and goals is concerned, i.e. they accept formulas from the language they are formalised in. However, the analysis of exemplary abductive problems revealed that the interpretation of logic programs in classical propositional logic, i.e. treating Horn clauses as classical implications vs Clark's completion, can change an abductive problem to become unsolvable or even leads to the situation, where it stops being a problem, because the abductive goal becomes obtainable from the knowledge base. Further comparison between those two systems requires generation and analysis of much larger collection of calculated examples and is included in the future work.

Finally, the abductive procedure that uses multi-criteria analysis along with sophisticated criteria to evaluate abductive hypotheses [30] was described. It is based on the classical propositional logic and consists of two phases: the first one concerns the generation of a set of abducibles by means of the Synthetic Tableaux Method [57]; the second one concerns the simplification of the generated set of abducibles and evaluation of abductive hypotheses by means of the mentioned multi-criteria analysis. As for the future work, it would be interesting to generate such abductive problems that would allow to compare both abductive procedures. This leads us to the comparison between both procedures. On the one hand, the difference between the procedure from [30] and the one proposed in this work is similar as in the case of ALP—the abductive procedure proposed in this work does not

require to generate a set of abducibles, but instead one abductive hypothesis is given as a result. On the other hand, the second difference is similar as in the case of AQAS, because it concerns the problem with the compatibility between the logic programs and the classical propositional logic.

Chapter 7
Conclusions and future work

The abductive procedure presented in this work is placed in a growing trend to combine advantages of the symbolic and neural frameworks. On the one hand, the formal framework of logic programs allows to precisely define used concepts, like for example abductive goal or abductive hypothesis. On the other hand, the ability of artificial neural networks to learn from examples gives possibility of creating structures that are very hard to grasp by means of strict algorithms or symbolic structures.

7.1 Summary

Let us recall the scheme of the abductive reasoning given in the Figure 1.3. The whole procedure begins when we are given a knowledge base in the form of a logic program and an abductive goal of the form of a Horn clause. Having the information about the properties of neural networks into which we can translate logic programs, especially that they model the way the immediate consequence operator works for logic programs, we can assume that constructing the definition of abductive goal in such a way that it depends on the immediate consequence operator can be helpful in further steps, like e.g. designing a training set for the neural networks. Therefore, the second Chapter (p. 11) is devoted for the formal description of logic programs and proving Theorem that allow to define an abductive goals in the form of Horn clauses in terms of the immediate consequence operator.

Moving according to arrow on the scheme we arrive at a neural network that is a result of the translation of a knowledge base in the form of a logic program along with an abductive goal in the form of a Horn clause. Before we describe the algorithm that allows for such translation, we have to precise the domain of the translation. Therefore, the third Chapter (p. 53) contains the description of neural networks as well as the notation used in further

part by the translation algorithm. Now we have described all corners of the scheme given in the Figure 1.3.

The next Chapter (p. 65) is devoted for the characterisation of all arrows that connect corners of the scheme. Firstly, the algorithm that translates logic programs into neural networks is defined as a modified version of Garcez's et al. [14] proposal. The introduced modifications were based on the results obtained in the second Chapter, especially the definition of abductive goal based on the immediate consequence operator and the theorems concerning the relation between the immediate consequence operator and the notion of a logical consequence in the form of a Horn clause of a logic program. Afterwards, the reverse translation was defined, i.e. from neural networks to logic programs. In addition, in order to reduce the set of Horn clauses resulting from this translation, a procedure of simplification for logic programs was defined. Having defined both translations we proceeded to the problem of neural networks training. There was only one training example for neural networks, which was defined on the basis of the way the translation algorithm from neural networks to logic programs work, and the properties of the theorem used to define abductive goals by means of the immediate consequence operator. Afterwards, we moved to the last element of the scheme from the Figure 1.3, which is the definition of the abductive hypothesis as a symmetric difference between the initial logic program and the one that is a result of the translation of the trained neural network. At this point there were also discussed criteria of evaluation of abductive hypotheses, where some of them were implemented in the proposed abductive procedure.

At this point the abductive procedure is fully defined. In the next two chapters the implementation of the abductive procedure is described along with exemplary results, and the comparison between a couple of other abductive procedures and the one proposed in this work is conveyed.

It should be underlined that apart from particular results described in different chapters the asset of this work lays also in the novelty of the approach and the way it is designed, i.e. how it combines knowledge that concerns formal systems like logic programs, artificial neural networks and reasoning.

7.2 Moving further

Every part of proposed abductive procedure yields ideas about the further development. Starting from the logic programs, there is an obvious direction in which this work should go, i.e. the extension of the procedure for extended logic programs, which allow to use both negations: the default and the classical one. Garcez et al. [14] proposed a translation algorithm from extended logic programs to neural networks. In addition, there are also other formal systems that can can be translated into neural networks, like e.g. in [4]. It was also mentioned in the Section 4.3 that abductive hypothesis can be defined in

a more sophisticated way that would allow to investigate not only additions and removals of Horn clauses, but also their changes, e.g. from a Horn clause that has a body to a fact.

As far as neural networks are concerned there are couple of possible ways of further development of the procedure. The wide-scale research of many different examples of abductive problems and with the use of all of the factors that takes as arguments the translation algorithm could lead to the identification of such neural networks characteristics that improve the formulation of "good" abductive hypotheses. Therefore, there are plans for the implementation of the evaluation of abductive hypotheses, as it was done e.g. in [30].

There was only one training set used in the examples described in the Section 5.2 that was made on the ground of the immediate consequence operator method of obtaining the minimal models for acceptable logic programs, and with the assumption that knowledge about good abductive hypotheses contained in the training example should be minimised. However, it could be the case that the training based on the more diverse set of training examples would yield significant improvement. In addition, there can be used different methods of neural networks training, e.g. instead of the backpopagation algorithm a genetic algorithms could be employed [38].

Finally, in the Chapter 6 the problem of comparability between the proposed procedure and abductive procedures based on the classical propositional logic was addressed. Solving it would open the way to convey wide-scale comparison between different abductive procedures, like for example AQAS [6] or the work of Komosiński et al. [30].

Chapter 8
Appendix

8.1 Logic programs obtained in experiments

8.1.1 Logic programs obtained for the second abductive problem

$$\mathcal{P}_2^1 = \{ \begin{array}{l} a_1 \leftarrow a_2 \\ a_1 \leftarrow a_3 \\ a_4 \leftarrow \\ a_3 \leftarrow \\ a_2 \leftarrow \\ a_5 \leftarrow \quad \} \end{array} \qquad \mathcal{P}_2^2 = \{ \begin{array}{l} a_1 \leftarrow a_2 \\ a_1 \leftarrow a_3 \\ a_4 \leftarrow a_5 \\ a_3 \leftarrow a_5 \\ a_2 \leftarrow a_3 \\ a_5 \leftarrow \quad \} \end{array} \qquad \mathcal{P}_2^3 = \{ \begin{array}{l} a_1 \leftarrow a_2 \\ a_1 \leftarrow a_3 \\ a_4 \leftarrow a_5 \\ a_3 \leftarrow \sim a_1, \sim a_4 \\ a_3 \leftarrow a_5, \sim a_5, \sim a_2 \\ a_3 \leftarrow a_5, \sim a_1 \\ a_3 \leftarrow \sim a_1, \sim a_2 \\ a_2 \leftarrow a_3 \\ a_5 \leftarrow \quad \} \end{array}$$

$$\mathcal{P}_2^4 = \{ \begin{array}{l} a_1 \leftarrow a_2 \\ a_1 \leftarrow a_3 \\ a_4 \leftarrow \sim a_1 \\ a_3 \leftarrow \sim a_1 \\ a_2 \leftarrow a_3 \\ a_5 \leftarrow \quad \} \end{array} \qquad \mathcal{P}_2^5 = \{ \begin{array}{l} a_1 \leftarrow a_2 \\ a_1 \leftarrow a_3 \\ a_4 \leftarrow a_5 \\ a_3 \leftarrow \\ a_2 \leftarrow \\ a_5 \leftarrow \quad \} \end{array} \qquad \mathcal{P}_2^6 = \{ \begin{array}{l} a_1 \leftarrow \sim a_5, a_3, \sim a_5 \\ a_1 \leftarrow a_2 \\ a_4 \leftarrow \\ a_3 \leftarrow \\ a_2 \leftarrow \\ a_5 \leftarrow \quad \} \end{array}$$

$$\mathcal{P}_2^7 = \{ \quad a_1 \leftarrow a_2 \quad a_1 \leftarrow a_3 \quad a_4 \leftarrow a_5, \sim a_3 \quad a_4 \leftarrow \sim a_1, \sim a_3, \sim a_2 \quad a_4 \leftarrow a_5, \sim a_1 \quad a_4 \leftarrow a_5, \sim a_2 \quad a_3 \leftarrow \sim a_1, \sim a_4 \quad a_3 \leftarrow a_5, \sim a_4 \quad a_3 \leftarrow \sim a_1, \sim a_2 \quad a_3 \leftarrow a_5, \sim a_1 \quad a_2 \leftarrow a_3 \quad a_5 \leftarrow \quad \}$$

$$\mathcal{P}_2^8 = \{ \quad a_1 \leftarrow a_2 \quad a_1 \leftarrow \sim a_4, a_3 \quad a_1 \leftarrow a_3, \sim a_5 \quad a_4 \leftarrow \quad a_3 \leftarrow \quad a_2 \leftarrow \quad a_5 \leftarrow \quad \}$$

$$\mathcal{P}_2^9 = \{ \quad a_1 \leftarrow a_2 \quad a_1 \leftarrow a_3, \sim a_5 \quad a_4 \leftarrow \quad a_3 \leftarrow \quad a_2 \leftarrow \quad a_5 \leftarrow \quad \}$$

$$\mathcal{P}_2^{10} = \{ \quad a_1 \leftarrow a_2 \quad a_1 \leftarrow a_3 \quad a_4 \leftarrow \sim a_1, a_3 \quad a_4 \leftarrow a_5, \sim a_1 \quad a_4 \leftarrow a_5, a_3, a_2 \quad a_4 \leftarrow a_2, \sim a_1 \quad a_3 \leftarrow \sim a_1 \quad a_2 \leftarrow a_3 \quad a_5 \leftarrow \quad \}$$

$$\mathcal{P}_2^{11} = \{ \quad a_1 \leftarrow a_2 \quad a_1 \leftarrow \sim a_5, a_3 \quad a_1 \leftarrow a_4, a_3 \quad a_4 \leftarrow \sim a_1, \sim a_3 \quad a_4 \leftarrow \sim a_1, a_5 \quad a_4 \leftarrow \sim a_1, \sim a_2 \quad a_4 \leftarrow a_5, \sim a_2 \quad a_3 \leftarrow \sim a_1, \sim a_2 \quad a_3 \leftarrow \sim a_1, a_4 \quad a_3 \leftarrow a_5, a_4, \sim a_2 \quad a_3 \leftarrow a_5, \sim a_1 \quad a_2 \leftarrow a_3 \quad a_2 \leftarrow \sim a_1, \sim a_5 \quad a_5 \leftarrow \quad \}$$

$$\mathcal{P}_2^{12} = \{ \quad a_1 \leftarrow a_2 \quad a_1 \leftarrow a_3 \quad a_4 \leftarrow a_5 \quad a_3 \leftarrow \sim a_4, \sim a_2 \quad a_3 \leftarrow a_5, \sim a_1 \quad a_3 \leftarrow \sim a_2, \sim a_1 \quad a_3 \leftarrow \sim a_4, \sim a_1 \quad a_2 \leftarrow a_3 \quad a_5 \leftarrow \quad \}$$

$$\mathcal{P}_2^{13} = \{ \quad a_1 \quad a_4 \leftarrow \quad a_3 \leftarrow \quad a_2 \leftarrow \quad a_5 \leftarrow \quad \}$$

$$\mathcal{P}_2^{14} = \{ \quad a_1 \leftarrow a_2 \quad a_1 \leftarrow a_3 \quad a_4 \leftarrow \quad a_3 \leftarrow \quad a_2 \leftarrow a_3 \quad a_5 \leftarrow \quad \}$$

$$\mathcal{P}_2^{15} = \{ \quad a_1 \leftarrow a_2 \quad a_1 \leftarrow a_3 \quad a_4 \leftarrow a_5 \quad a_3 \leftarrow \sim a_1, \sim a_4 \quad a_3 \leftarrow a_5, \sim a_4 \quad a_3 \leftarrow \sim a_1, \sim a_2 \quad a_3 \leftarrow a_5, \sim a_1 \quad a_2 \leftarrow a_3 \quad a_5 \leftarrow \quad \}$$

$$\mathcal{P}_2^{16} = \{ \quad a_1 \leftarrow a_2, a_3 \quad a_1 \leftarrow \sim a_4, a_3, \sim a_5 \quad a_1 \leftarrow \sim a_4, a_2 \quad a_1 \leftarrow \sim a_5, a_2 \quad a_4 \leftarrow a_5 \quad a_3 \leftarrow \sim a_4, \sim a_2 \quad a_3 \leftarrow \sim a_1, a_5 \quad a_3 \leftarrow \sim a_1, \sim a_2 \quad a_3 \leftarrow \sim a_1, \sim a_4 \quad a_2 \leftarrow a_3 \quad a_5 \leftarrow \quad \}$$

$$\mathcal{P}_2^{17} = \{ \quad a_1 \leftarrow a_2 \quad a_1 \leftarrow a_3 \quad a_4 \leftarrow a_5, \sim a_3 \quad a_4 \leftarrow \sim a_1, \sim a_3, \sim a_2 \quad a_4 \leftarrow \sim a_1, a_5 \quad a_4 \leftarrow a_5, \sim a_2 \quad a_3 \leftarrow \sim a_1, \sim a_4 \quad a_3 \leftarrow \sim a_4, a_5, \sim a_2 \quad a_3 \leftarrow \sim a_1, \sim a_2 \quad a_3 \leftarrow a_5, \sim a_1 \quad a_2 \leftarrow a_3, \sim a_5 \quad a_2 \leftarrow a_3, \sim a_1 \quad a_2 \leftarrow a_3, \sim a_5 \quad a_5 \leftarrow \quad \}$$

$$\mathcal{P}_2^{18} = \{ \quad a_1 \leftarrow a_2, a_5 \quad a_1 \leftarrow a_2, \sim a_5 \quad a_1 \leftarrow a_3 \quad a_4 \quad a_3 \quad a_2 \quad a_5 \leftarrow \quad \}$$

$$\mathcal{P}_2^{19} = \{ \quad a_1 \leftarrow a_2 \quad a_1 \leftarrow a_3 \quad a_4 \leftarrow \sim a_1, \sim a_2 \quad a_4 \leftarrow \sim a_1, a_3 \quad a_4 \leftarrow a_5, a_3, \sim a_2 \quad a_4 \leftarrow a_5, \sim a_1 \quad a_3 \leftarrow \sim a_1, \sim a_2 \quad a_3 \leftarrow \sim a_1, a_4 \quad a_3 \leftarrow a_5, a_4, \sim a_2 \quad a_3 \leftarrow a_5, \sim a_1 \quad a_2 \leftarrow a_3 \quad a_5 \leftarrow \quad \}$$

$$\mathcal{P}_2^{20} = \{ \quad a_1 \leftarrow a_2 \quad a_1 \leftarrow a_3 \quad a_4 \leftarrow a_5 \quad a_3 \leftarrow \sim a_4, \sim a_2 \quad a_3 \leftarrow \sim a_1, \sim a_2 \quad a_3 \leftarrow \sim a_1, \sim a_4 \quad a_2 \leftarrow a_3 \quad a_2 \leftarrow \sim a_1, \sim a_4, \sim a_5 \quad a_5 \leftarrow \quad \}$$

$$\mathcal{P}_2^{21} = \{ \quad a_1 \leftarrow a_2, a_5 \quad a_1 \leftarrow a_3 \quad a_4 \leftarrow \sim a_1 \quad a_4 \leftarrow a_5, a_3, \sim a_2 \quad a_3 \leftarrow \sim a_1, \sim a_2 \quad a_3 \leftarrow \sim a_1, a_4 \quad a_3 \leftarrow a_5, a_4, \sim a_2 \quad a_3 \leftarrow a_5, \sim a_1 \quad a_2 \leftarrow a_3 \quad a_5 \leftarrow \quad \}$$

$$\mathcal{P}_2^{22} = \{ \quad a_1 \leftarrow a_2 \quad a_1 \leftarrow a_3 \quad a_4 \leftarrow a_5, a_2 \quad a_4 \leftarrow \sim a_1, a_3 \quad a_4 \leftarrow a_5, \sim a_1 \quad a_4 \leftarrow \sim a_1, a_2 \quad a_3 \leftarrow a_5, \sim a_4, a_2 \quad a_3 \leftarrow \sim a_1 \quad a_2 \leftarrow a_3 \quad a_5 \leftarrow \quad \}$$

$$\mathcal{P}_2^{23} = \{ \quad a_1 \leftarrow a_2 \quad a_1 \leftarrow a_3 \quad a_4 \leftarrow \quad a_3 \leftarrow \sim a_4, \sim a_2 \quad a_3 \leftarrow \sim a_1, \sim a_2 \quad a_3 \leftarrow \sim a_1, \sim a_4 \quad a_2 \leftarrow a_3 \quad a_5 \leftarrow \quad \}$$

$$\mathcal{P}_2^{24} = \{ \quad a_1 \leftarrow a_2 \quad a_1 \leftarrow a_3 \quad a_4 \leftarrow a_5 \quad a_4 \leftarrow \sim a_1, \sim a_2 \quad a_3 \leftarrow a_5 \quad a_3 \leftarrow \sim a_1, \sim a_2 \quad a_2 \leftarrow a_3 \quad a_3 \leftarrow a_5, \sim a_1 \quad a_5 \leftarrow \quad \}$$

$$\mathcal{P}_2^{25} = \{ \quad a_1 \leftarrow a_2 \quad a_1 \leftarrow a_3 \quad a_4 \leftarrow a_5 \quad a_4 \leftarrow \sim a_1, \sim a_2 \quad a_3 \leftarrow \sim a_1, a_4, \sim a_2 \quad a_3 \leftarrow a_5 \quad a_2 \leftarrow a_3$$

$$\mathcal{P}_2^{26} = \{ \quad a_1 \leftarrow a_2 \quad a_1 \leftarrow a_3 \quad a_4 \leftarrow a_5 \quad a_4 \leftarrow \sim a_1, \sim a_2 \quad a_3 \leftarrow \quad a_2 \leftarrow a_3 \quad a_5 \leftarrow \quad \}$$

$$\mathcal{P}_2^{27} = \{ \quad a_1 \leftarrow a_2 \quad a_1 \leftarrow a_3 \quad a_4 \leftarrow a_5 \quad a_4 \leftarrow \sim a_1, \sim a_3, \sim a_2 \quad a_3 \leftarrow a_5 \quad a_2 \leftarrow a_3 \quad a_5 \leftarrow \quad \}$$

$$\mathcal{P}_2^{28} = \{ \quad a_1 \leftarrow a_2 \quad a_1 \leftarrow a_3 \quad a_4 \leftarrow a_5 \quad a_3 \leftarrow a_5 \quad a_3 \leftarrow \sim a_1, \sim a_4, \sim a_2 \quad a_2 \leftarrow a_3 \quad a_5 \leftarrow \quad \}$$

$$\mathcal{P}_2^{29} = \{ \quad a_1 \leftarrow a_2 \quad a_1 \leftarrow a_3 \quad a_4 \leftarrow a_3, a_2 \quad a_4 \leftarrow a_5 \quad a_4 \leftarrow a_1, a_2 \quad a_4 \leftarrow a_1, a_3 \quad a_3 \leftarrow \quad a_2 \leftarrow \quad a_5 \leftarrow \quad \}$$

$$\mathcal{P}_2^{30} = \{ \quad a_1 \leftarrow a_2 \quad a_1 \leftarrow a_3 \quad a_4 \leftarrow a_5 \quad a_3 \leftarrow \quad a_2 \leftarrow a_3 \quad a_5 \leftarrow \quad \}$$

$$\mathcal{P}_2^{31} = \{ \quad a_1 \leftarrow a_2 \quad a_1 \leftarrow a_3 \quad a_4 \leftarrow \quad a_3 \leftarrow a_5 \quad a_2 \leftarrow a_3 \quad a_5 \leftarrow \quad \}$$

$$\mathcal{P}_2^{32} = \{ \quad a_1 \leftarrow a_2 \quad a_1 \leftarrow a_3 \quad a_4 \leftarrow \quad a_3 \leftarrow a_5 \quad a_2 \leftarrow a_3 \quad a_5 \leftarrow \quad \}$$

$$\mathcal{P}_2^{33} = \{ \quad a_1 \leftarrow a_2 \quad a_1 \leftarrow a_3 \quad a_4 \leftarrow \quad a_3 \leftarrow a_5 \quad a_2 \leftarrow \quad a_5 \leftarrow \quad \}$$

$$\mathcal{P}_2^{34} = \{ \quad a_1 \leftarrow a_2 \quad a_1 \leftarrow a_3 \quad a_4 \leftarrow \quad a_3 \leftarrow a_4, a_2 \quad a_3 \leftarrow a_5 \quad a_3 \leftarrow a_1 \quad a_2 \leftarrow a_3 \quad a_5 \leftarrow \quad \}$$

$$\mathcal{P}_2^{35} = \{ \quad a_1 \leftarrow a_2 \quad a_1 \leftarrow a_3 \quad a_4 \leftarrow a_5 \quad a_3 \leftarrow a_5 \quad a_3 \leftarrow a_1, a_4, a_2 \quad a_2 \leftarrow a_3 \quad a_5 \leftarrow \quad \}$$

$$\mathcal{P}_2^{36} = \{ \quad a_1 \leftarrow a_2 \quad a_1 \leftarrow a_3 \quad a_4 \leftarrow a_5 \quad a_4 \leftarrow a_2 \quad a_4 \leftarrow a_3 \quad a_4 \leftarrow a_1 \quad a_3 \leftarrow \quad a_2 \leftarrow a_3 \quad a_5 \leftarrow \quad \}$$

$$\mathcal{P}_2^{37} = \{ \quad a_1 \leftarrow a_2 \quad a_1 \leftarrow a_3 \quad a_4 \leftarrow \quad a_3 \leftarrow a_5 \quad a_3 \leftarrow a_1, a_4, a_2 \quad a_2 \leftarrow a_3 \quad a_5 \leftarrow \quad \}$$

$$\mathcal{P}_2^{38} = \{ \quad a_1 \leftarrow a_2 \quad a_1 \leftarrow a_3 \quad a_4 \leftarrow a_5 \quad a_4 \leftarrow a_1, a_3, a_2 \quad a_3 \leftarrow a_5 \quad a_2 \leftarrow a_3 \quad a_5 \leftarrow \quad \}$$

$$\mathcal{P}_2^{39} = \{ \quad a_1 \leftarrow a_2 \quad a_1 \leftarrow a_3 \quad a_4 \leftarrow a_5 \quad a_3 \leftarrow a_5 \quad a_2 \leftarrow a_3 \quad a_2 \leftarrow a_1, a_4, a_5 \quad a_5 \leftarrow \quad \}$$

$$\mathcal{P}_2^{40} = \{ \quad a_1 \leftarrow a_2 \quad a_1 \leftarrow a_3 \quad a_4 \leftarrow a_3, a_2 \quad a_4 \leftarrow a_5 \quad a_4 \leftarrow a_1, a_2 \quad a_4 \leftarrow a_1, a_3 \quad a_3 \leftarrow a_5 \quad a_3 \leftarrow \quad a_2 \leftarrow a_3 \quad a_5 \leftarrow \quad \}$$

8.1.2 Logic programs obtained for the third abductive problem

$$\mathcal{P}_2^{1} = \{ \quad a_1 \leftarrow a_2 \quad a_1 \leftarrow a_3 \quad a_4 \leftarrow a_1 \quad a_3 \leftarrow \quad a_2 \leftarrow \quad a_5 \leftarrow \quad \}$$

$$\mathcal{P}_2^{2} = \{ \quad a_1 \leftarrow a_2 \quad a_1 \leftarrow a_3 \quad a_4 \leftarrow a_5 \quad a_3 \leftarrow \quad a_2 \leftarrow \quad a_5 \leftarrow \quad \}$$

$$\mathcal{P}_2^{3} = \{ \quad a_1 \leftarrow a_2 \quad a_1 \leftarrow a_3 \quad a_4 \leftarrow a_5 \quad a_4 \leftarrow \sim a_1, a_3 \quad a_3 \leftarrow \quad a_2 \leftarrow \quad a_5 \leftarrow \quad \}$$

$$\mathcal{P}_2^4 = \{ \begin{array}{l} a_1 \leftarrow a_2 \\ a_1 \leftarrow a_3 \\ a_4 \leftarrow \\ a_3 \leftarrow \\ a_2 \leftarrow \\ a_5 \leftarrow \quad \} \end{array}$$

$$\mathcal{P}_2^5 = \{ \begin{array}{l} a_1 \leftarrow a_2 \\ a_1 \leftarrow a_3 \\ a_4 \leftarrow a_5 \\ a_3 \leftarrow a_5 \\ a_2 \leftarrow \\ a_5 \leftarrow \quad \} \end{array}$$

$$\mathcal{P}_2^6 = \{ \begin{array}{l} a_1 \leftarrow a_2 \\ a_1 \leftarrow a_3 \\ a_4 \leftarrow \\ a_3 \leftarrow a_5 \\ a_2 \leftarrow \\ a_5 \leftarrow \quad \} \end{array}$$

$$\mathcal{P}_2^7 = \{ \begin{array}{l} a_1 \leftarrow a_2 \\ a_1 \leftarrow a_3 \\ a_4 \leftarrow a_5 \\ a_4 \leftarrow a_3 \\ a_4 \leftarrow \sim a_1 \\ a_3 \leftarrow \\ a_2 \leftarrow \\ a_5 \leftarrow \quad \} \end{array}$$

$$\mathcal{P}_2^8 = \{ \begin{array}{l} a_1 \leftarrow a_2 \\ a_1 \leftarrow a_3 \\ a_4 \leftarrow a_5 \\ a_3 \leftarrow \sim a_1, a_4, \sim a_2 \\ a_3 \leftarrow a_5 \\ a_2 \leftarrow \\ a_5 \leftarrow \quad \} \end{array}$$

$$\mathcal{P}_2^9 = \{ \begin{array}{l} a_1 \leftarrow a_2 \\ a_1 \leftarrow a_3 \\ a_4 \leftarrow \\ a_3 \leftarrow a_4, \sim a_2 \\ a_3 \leftarrow \sim a_1, \sim a_2 \\ a_3 \leftarrow a_5 \\ a_2 \leftarrow \\ a_5 \leftarrow \quad \} \end{array}$$

$$\mathcal{P}_2^{10} = \{ \begin{array}{l} a_1 \leftarrow a_2 \\ a_1 \leftarrow a_3 \\ a_4 \leftarrow a_5 \\ a_3 \leftarrow \sim a_2 \\ a_3 \leftarrow a_5 \\ a_3 \leftarrow \sim a_1 \\ a_3 \leftarrow a_4 \\ a_2 \leftarrow \\ a_5 \leftarrow \quad \} \end{array}$$

8.2 The detailed translation algorithm $T_{\mathcal{P}\to\mathfrak{N}}$

Algorithm 1 Translation from $\mathcal{P}$ to $\mathfrak{N}$

Input: $\mathcal{P}$, l, $A^f_{\min}$, W^f, β, b_a, r
Output: $\mathfrak{N}$

$A^b_{\min} \leftarrow \dfrac{\max(K_{\mathcal{P}} \cup Mu_{\mathcal{P}}) - 1}{\max(K_{\mathcal{P}} \cup Mu_{\mathcal{P}}) + 1}$
$A_{\min} \leftarrow A^b_{\min} + A^f_{\min}$
$W^b_1 \leftarrow -\dfrac{2}{\beta} \cdot \dfrac{\ln(1 - A_{\min}) - \ln(1 + A_{\min})}{\max(K_{\mathcal{P}} \cup Mu_{\mathcal{P}})(A_{\min} - 1) + A_{\min} + 1}$
$W^b_2 \leftarrow -\dfrac{2}{\beta} \cdot \dfrac{\ln(1 - A_{\min}) - \ln(1 + A_{\min}) + r(l+1)}{\max(Mu_{\mathcal{P}})(A_{\min} - 1) + A_{\min} + 1}$
if $W^b_1 > W^b_2$ **then**
    $W^b \leftarrow W^b_1$
else
    $W^b \leftarrow W^b_2$
end if

$W \leftarrow W^b + W^f$

$ovl \leftarrow$ GETOVERLAPPING($\mathcal{P}$)

$baseNinfo \leftarrow$ BASICGEN($A_{\min}$, W, $\mathcal{P}$, β, b_a, ovl)
$\mathcal{N}_I \leftarrow baseNinfo[0]$
$\mathcal{N}_H \leftarrow baseNinfo[1]$
$\mathcal{N}_O \leftarrow baseNinfo[2]$
$\mathcal{C}_{i\to h} \leftarrow baseNinfo[3]$
$\mathcal{C}_{h\to o} \leftarrow baseNinfo[4]$

$additionalNinfo \leftarrow$ ADDGEN(l, $A_{\min}$, β, b_a, $\mathcal{P}$, $\mathcal{N}_I$, $\mathcal{N}_H$, $\mathcal{N}_O$, $\mathcal{C}_{i\to h}$, $\mathcal{C}_a$)
$\mathcal{N}_H \leftarrow \mathcal{N}_H \cup additionalNinfo[0]$
$\mathcal{C}_a \leftarrow additionalNinfo[1]$

$recursiveNinfo \leftarrow$ ADDRECURSIVE($\mathcal{N}_I$, $\mathcal{N}_O$, $\mathcal{N}_R$, ovl)
$\mathcal{C}_r \leftarrow recursiveNinfo[0]$
$\mathcal{N}_r \leftarrow recursiveNinfo[1]$

Algorithm 2 Function that generates the set of atoms and their counterparts

```
function getOverlapping(𝒫)
    base ← B_𝒫
    atoms ← ∅

    repeat
        atom ← base[0]

        if m({atom}, h) ∈ B_𝒫 or um({atom}, h) ∈ B_𝒫 then
            atoms ← atoms ∪ {atom}
        end if

        base ← base \ {atom}
    until base = ∅

    return atoms
end function
```

Algorithm 3 Function for "base" neural network (part 1)

```
function BASICGEN(A_min, W, 𝒫, β, b_a, ovl)
    clauses ← 𝒫
    𝒩_I ← ∅
    𝒩_H ← ∅
    𝒩_O ← ∅
    𝒞_{i→h} ← ∅
    𝒞_{h→o} ← ∅
    i ← 1
    j ← 1
    k ← 1
    𝒩_I ← 𝒩_I ∪ {𝔱}
    𝒩_H ← 𝒩_H ∪ {𝔥_ι}

    repeat
        cl ← clauses[0]
        neuO ← GETNEURON(𝒩_O, head(cl))

        if body(cl) = ∅ then
            if neuO = 0 then
                if head(cl) ∉ ovl then
                    𝔦_i ← ⟨g(·) , 0 , head(cl) , A_min⟩
                    i ← i + 1
                    𝒩_I ← 𝒩_I ∪ {𝔦_i}
                end if

                b_o ← ((1 + A_min)(1 − μ(cl)))/2 W
                𝔬_k ← ⟨h(·) , b_o , head(cl) , A_min⟩
                k ← k + 1
                𝒩_O ← 𝒩_O ∪ {𝔬_k}
                𝒞_{h→o} ← 𝒞_{h→o} ∪ {⟨𝔥_ι, 𝔬_k⟩}
            else
                𝒞_{h→o} ← 𝒞_{h→o} ∪ {⟨𝔥_ι, neuO⟩}
            end if
        else
            b_h ← ((1 + A_min)(1 − 𝔨(cl)))/2 W
            𝔥_j ← ⟨h(·) , b_h , n_j , A_min⟩
            j ← j + 1
            𝒩_H ← 𝒩_H ∪ {𝔥_j}

            if neuO = 0 then
                𝔦_i ← ⟨g(·) , 0 , head(cl) , A_min⟩
                i ← i + 1
                𝒩_I ← 𝒩_I ∪ {𝔦_i}
                b_o ← ((1 + A_min)(1 − μ(cl)))/2 W
                𝔬_k ← ⟨h(·) , b_o , head(cl) , A_min⟩
                k ← k + 1
                𝒩_O ← 𝒩_O ∪ {𝔬_k}
                𝒞_{h→o} ← 𝒞_{h→o} ∪ {⟨𝔥_j, 𝔬_k⟩}
            else
                𝒞_{h→o} ← 𝒞_{h→o} ∪ {⟨𝔥_j, neuO⟩}

                if head(cl) ∈ body(cl) then
                    neuI ← GETNEURON(𝒩_I, head(cl))
                    𝒞_{i→h} ← 𝒞_{i→h} ∪ {⟨neuI, 𝔥_j⟩}
                end if
            end if
```

Algorithm 4 Function for "base" neural network (part 2)

```
52:             literals ← body(cl) \ {head(cl)}

53:             repeat
54:                 lit ← literals[0]
55:                 neuI ← getNeuron(N_I, lit)

56:                 if neuI = 0 then
57:                     i_i ← ⟨g(·) , 0 , lit , A_min⟩
58:                     i ← i + 1
59:                     C_{i→h} ← C_{i→h} ∪ {⟨i_i, h_j⟩}
60:                 else
61:                     C_{i→h} ← C_{i→h} ∪ {⟨i_i, neuI⟩}
62:                 end if

63:                 neuO ← getNeuron(N_O, lit)

64:                 if neuO = 0 then
65:                     if lit ∉ B^h_P then
66:                         o_k ← ⟨h(·) , b_a , lit , A_min⟩
67:                         k ← k + 1
68:                     else
69:                         headCl ← getClause(P, lit)
70:                         b_o ← (1 + A_min)(1 − μ(headCl)) / 2 W
71:                         o_k ← ⟨h(·) , b_o , lit , A_min⟩
72:                         k ← k + 1
73:                     end if
74:                 end if

75:                 literals ← literals \ {lit}
76:             until literals = ∅
77:         end if

78:         clauses ← clauses \ {cl}
79:     until clauses = ∅

80:     return [N_I, N_H, N_O, C_{i→h}, C_{h→o}]
81: end function
```

Algorithm 5 Function for checking if there is a neuron associated with a given atom

```
1: function getNeuron(set, atom)
2:     repeat
3:         neu ← set[0]
4:         if neu[2] = atom then
5:             return neu
6:         end if
7:         set ← set \ {neu}
8:     until set = ∅
9:     return 0
10: end function
```

Algorithm 6 Function for additional neurons and additional connections generation

```
function ADDGEN(l, A_min, β, b_a, P, N_I, N_H, N_O, C_{i→h}, C_a)
    outputN ← N_O
    i ← c(N_H) + 1
    z ← 1
    facts ← GETFACTS(P)

    repeat
        out ← outputN[0]

        for x ← 1, l do
            if out[2] ∉ facts then
                ℌ_i ← ⟨h(·) , b_a , a_z , A_min⟩
                i ← i + 1
                z ← z + 1
                x ← x + 1
                N_H ← N_H ∪ {ℌ_i}
                C_a ← C_a ∪ {c_{ℌ_i out}}
            end if
        end for

        outputN ← outputN \ {out}
    until outputN = ∅

    inputN ← N_I
    hiddenN ← N_H

    repeat
        inp ← inputN[0]

        repeat
            hid ← hiddenN[0]

            if ⟨inp, hid⟩ ∉ C_{i→h} and inp ≠ t and hid ≠ ℌ_t then
                out ← GETNEURON(N_O, inp[2])

                if ⟨hid, out⟩ ∉ C_{h→o} and ⟨hid, out⟩ ∉ C_a then
                    C_a ← C_a ∪ {⟨inp hid⟩}
                end if
            end if

            hiddenN ← hiddenN \ {hid}
        until hiddenN = ∅

        inputN ← inputN \ {inp}
    until inputN = ∅

    return [N_H, C_a]
end function
```

Algorithm 7 Function for checking if there is a Horn clause that has a given atom as head in a logic program $\mathcal{P}$

```
function getClause(P, lit)
    clauses ← P
    repeat
        cl ← clauses[0]

        if head(cl) = lit then
            return cl
        end if

        clauses ← clauses \ {cl}
    until clauses = ∅

    return 0
end function
```

Algorithm 8 Function for collecting all facts from a logic program $\mathcal{P}$

```
function getFacts(P)
    clauses ← P
    facts ← ∅
    repeat
        cl ← clauses[0]

        if body(cl) = ∅ then
            facts ← facts ∪ {head(cl)}
        end if

        clauses ← clauses \ {cl}
    until clauses = ∅

    return facts
end function
```

Algorithm 9 Function for checking if in a given set of neurons there is a neuron that is associated with a counterpart for the atom associated with a given neuron

```
function getComp(neuSet, neuron)
    neurons ← neuSet
    repeat
        neu ← neurons[0]

        if neuron[2] = m({neu[2]}, h) or um({neuron[2]}, h) = neu[2] then
            return neu
        end if

        neurons ← neurons \ {neu}
    until neurons = ∅

    return 0
end function
```

Algorithm 10 Function for creating additional recursive neurons and recursive connections

```
function addRecursive(𝒩_I, 𝒩_O, 𝒩_r, ovl)
    outputN ← 𝒩_O
    i ← 1

    repeat
        out ← outputN[0]

        if out[2] ∉ ovl then
            inp ← getNeuron(𝒩_I, out[2])
            𝒞_r ← 𝒞_r ∪ {⟨out, inp⟩}
        else
            𝔯_i ← ⟨k(·), 0, oi_i, 0⟩
            i ← i + 1
            comp ← getComp(𝒩_O, out)
            𝒞_r ← 𝒞_r ∪ {⟨out, 𝔯_i⟩} ∪ {⟨comp, 𝔯_i⟩}
            inp ← getNeuron(𝒩_I, out[2])

            if inp = 0 then
                inpComp ← getNeuron(𝒩_I, comp[2])
                𝒞_r ← 𝒞_r ∪ {⟨𝔯_i, inpComp⟩}
            else
                𝒞_r ← 𝒞_r ∪ {⟨𝔯_i, inp⟩}
            end if

            outputN ← outputN \ {comp}
        end if

        outputN ← outputN \ {out}
    until outputN = ∅

    return ⟨𝒞_r, 𝒩_r⟩
end function
```

References

1. Atocha Aliseda. *Abductive Reasoning*, volume 330 of *Synthese Library*. Springer Netherlands, 2006.
2. Robert Andrews, Joachim Diederich, and Alan Tickle. Survey and critique of techniques for extracting rules from trained artificial neural networks. *Knowledge-based systems*, 8(6):373–389, 1995.
3. Krzysztof Apt and Dino Pedreschi. Reasoning about termination of pure prolog programs. *Information and computation*, 106(1):109–157, 1993.
4. Tarek R. Besold, Artur d'Avila Garcez, Keith Stenning, Leendert van der Torre, and Michiel van Lambalgen. Reasoning in non-probabilistic uncertainty: Logic programming and neural-symbolic computing as examples. *Minds and Machines*, 27(1):37–77, Mar 2017.
5. Yves Chauvin and David E. Rumelhart. *Backpropagation: theory, architectures, and applications*. Psychology Press, 1995.
6. Szymon Chlebowski and Andrzej Gajda. Abductive question-answer system (aqas) for classical propositional logic. In H. Christiansen, H. Jaudoin, P. Chountas, T. Andreasen, and H. L. Larsen, editors, *Proceedings of Flexible Query Answering Systems—12th International Conference*, volume 10333, pages 3–14. Lecture Notes in Computer Science, 2017.
7. Keith L. Clark. Negation as failure. In *Logic and data bases*, pages 293–322. Springer, 1978.
8. Gerald Edelman. Neural darwinism: selection and reentrant signaling in higher brain function. *Neuron*, 10(2):115–125, 1993.
9. Kave Eshghi and Robert Kowalski. Abduction compared with negation by failure. In *Proceedings of the 6th International Conference on Logic Programming*, volume 89, pages 234–255, Lisbon, 1989.
10. Melvin Fitting. *First-order logic and automated theorem proving*. Springer Science & Business Media, 2012.
11. Li-Min Fu and Li-Chen Fu. Mapping rule-based systems into neural architecture. *Knowledge-Based Systems*, 3(1):48–56, 1990.
12. Dov M. Gabbay and John Woods. *The Reach of Abduction. Insight and Trial*. Elsevier, London, 2005.
13. Andrzej Gajda, Adam Kups, and Mariusz Urbański. A connectionist approach to abductive problems: employing a learning algorithm. In M. Ganzha, L. Maciaszek, and M. Paprzycki, editors, *Proceedings of the 2016 Federated Conference on Computer Science and Information Systems*, volume 8, pages 353–362. IEEE, Annals of Computer Science and Information Systems, 2016.

14. Artur S. d'Avila Garcez, Krysia Broda, and Dov M. Gabbay. *Neural-Symbolic Learning Systems: Foundations and Applications*. Springer-Verlag, London, 2002.
15. Artur S. d'Avila Garcez, Dov M. Gabbay, Oliver Ray, and John Woods. Abductive reasoning in neural-symbolic systems. *Topoi: An International Review of Philosophy*, 26(1):37–49, 2007.
16. Artur S. d'Avila Garcez, Alessandra Russo, Bashar Nuseibeh, and Jeff Kramer. Combining abductive reasoning and inductive learning to evolve requirements specifications. In *Software, IEE Proceedings*, volume 150, pages 25–38. IET, 2003.
17. Artur S. dAvila Garcez, Luís C. Lamb, Krysia Broda, and Dov M. Gabbay. Applying connectionist modal logics to distributed knowledge representation problems. *International Journal on Artificial Intelligence Tools*, 13(1):115–139, 2004.
18. Artur S. dAvila Garcez, Luís C. Lamb, and Dov M. Gabbay. Connectionist modal logic: Representing modalities in neural networks. *Theoretical Computer Science*, 371(1):34–53, 2007.
19. Peter Gärdenfors. *Belief Revision*. Tracts in Theoretical Computer Science 29. Cambridge University Press, 2003.
20. Michael Gelfond and Vladimir Lifschitz. The stable model semantics for logic programming. In *Proceedings of the 5th International Conference and Symposium on Logic Programming*, volume 88, pages 1070–1080, 1988.
21. Michael Gelfond and Vladimir Lifschitz. Classical negation in logic programs and disjunctive databases. *New generation computing*, 9(3–4):365–385, 1991.
22. Hayit Greenspan, Rodney Goodman, and Rama Chellappa. Combined neural network and rule-based framework for probabilistic pattern recognition and discovery. In *Advances in Neural Information Processing Systems*, pages 444–451, 1992.
23. Yoichi Hayashi. A neural expert system with automated extraction of fuzzy if-then rules and its application to medical diagnosis. In R. P. Lippman, J. E. Moody, and D. S. Touretzky, editors, *Advances in neural information processing systems*, pages 578–584, 1991.
24. Simon S. Haykin. *Neural networks and learning machines*. Pearson, New Jersey, 3rd edition, 2009.
25. John Hertz, Anders Krogh, and Richard G. Palmer. *Introduction to the theory of neural computation*, volume 1. Basic Books, 1991.
26. Mélanie Hilario. An overview of strategies for neurosymbolic integration. In Ron Sun and Frederic Alexandre, editors, *Connectionist-Symbolic Integration: From Unified to Hybrid Approaches*, chapter 2, pages 13–36. Psychology Press, 2013.
27. Steffen Hölldobler and Franz Kurfess. Chcl—a connectionist inference system. In *Parallelization in Inference Systems*, pages 318–342. Springer, 1992.
28. Antonis Kakas, Robert Kowalski, and Francesca Toni. Abductive logic programming. *Journal of Logic and Computation*, 2(6):719–770, 1992.
29. Antonis Kakas and Paolo Mancarella. Generalized stable models: A semantics for abduction. In *Proceedings of 9th European Conference on Artificial Intelligence, ECAI '90*, volume 90, pages 385–391, 1990.
30. Maciej Komosinski, Adam Kups, Dorota Leszczyńska-Jasion, and Mariusz Urbański. Identifying efficient abductive hypotheses using multi-criteria dominance relation. *ACM Transactions on Computational Logic*, 15(4), 2014.
31. Maciej Komosinski, Adam Kups, and Mariusz 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.
32. Maciej Komosinski and Szymon Ulatowski. Framsticks web site, 2016. `http://www.framsticks.com`.

33. Srinivas Krovvidy and William Wee. Hybrid architectures for intelligent systems. chapter An Intelligent Hybrid System for Wastewater Treatment, pages 357–377. CRC Press, Inc., Boca Raton, FL, USA, 1992.
34. Dorota Leszczyńska-Jasion. Socratic proofs for some normal modal propositional logics. *Logique et Analyse*, 47(185–188):259–285, 2004.
35. John W. Lloyd. *Foundations of logic programming*. Springer-Verlag, Berlin, 1993.
36. Edward McCluskey. Minimization of boolean functions. *The Bell System Technical Journal*, 35(6):1417–1444, 1956.
37. Warren McCulloch and Walter Pitts. A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, 5(4):115–133, 1943.
38. David J. Montana and Lawrence Davis. Training feedforward neural networks using genetic algorithms. In *Proceedings of the 11th International Joint Conference on Artificial Intelligence*, IJCAI'89, pages 762–767, San Francisco, CA, USA, 1989. Morgan Kaufmann Publishers Inc.
39. Mehmet A. Orgun and Wanli Ma. An overview of temporal and modal logic programming. In Dov M. Gabbay and Hans Jürgen Ohlbach, editors, *Temporal Logic*, pages 445–479, Berlin, Heidelberg, 1994. Springer Berlin Heidelberg.
40. Charles S. Peirce. *Collected Works*. Harvard University Press, Cambridge, MA., 1931–1958.
41. R Pfeifer, Z Schreter, F Fogelman-Soulie, and L Steels. Problem solving and reasoning: A connectionist perspective. *Connectionism in Perspective*, pages 229–243, 1989.
42. Willard Quine. The problem of simplifying truth functions. *The American Mathematical Monthly*, 59(8):521–531, 1952.
43. Oliver Ray and Artur S. dAvila Garcez. Towards the integration of abduction and induction in artificial neural networks. In *Proceedings of 2nd International Workshop on Hybrid Learning and Neural Networks*, 2006.
44. George Reeke, Leif Finkel, Olaf Sporns, and Gerald Edelman. Synthetic neural modeling: a multilevel approach to the analysis of brain complexity. *Signal and sense: Local and global order in perceptual maps*, pages 607–707, 1990.
45. George Reeke, Olaf Sporns, and Gerald Edelman. Synthetic neural modeling: the'darwin'series of recognition automata. *Proceedings of the IEEE*, 78(9):1498–1530, 1990.
46. Frank Rosenblatt. The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review*, 65(6):386, 1958.
47. David Rumelhart, Geoffrey Hinton, and Ronald Williams. Learning internal representations by error propagation. Technical report, California Univ San Diego La Jolla Inst for Cognitive Science, 1985.
48. David Rumelhart, Geoffrey Hinton, and Ronald Williams. Learning representations by back-propagating errors. *Nature*, 323(6088):533–536, 1986.
49. Alessandra Russo, Rob Miller, Bashar Nuseibeh, and Jeff Kramer. An abductive approach for handling inconsistencies in scr specifications. In *Proceedings of the Workshop on Intelligence Software Engineering (WISE), collocated with the International Conference on Software Engineering (ICSE)*, 2000.
50. Alessandra Russo, Rob Miller, Bashar Nuseibeh, and Jeff Kramer. An abductive approach for analysing event-based requirements specifications. In *International Conference on Logic Programming*, pages 22–37. Springer, 2002.
51. Yasubumi Sakakibara. Programming in modal logic: An extension of prolog based on modal logic. In Eiiti Wada, editor, *Logic Programming '86*, pages 81–91, Berlin, Heidelberg, 1987. Springer Berlin Heidelberg.
52. Tariq Samad. Hybrid distributed/local connectionist architectures. *Hybrid Architectures for Intelligent Systems*, pages 200–218, 1992.
53. David Touretzky. Boltzcons: Dynamic symbol structures in a connectionist network. *Artificial Intelligence*, 46(1-2):5–46, 1990.

54. David Touretzky and Geoffrey Hinton. A distributed connectionist production system. *Cognitive science*, 12(3):423–466, 1988.
55. Geoffrey Towell and Jude Shavlik. Knowledge-based artificial neural networks. *Artificial intelligence*, 70(1–2):119–165, 1994.
56. Mariusz Urbański. Remarks on synthetic tableaux for classical propositional calculus. *Bulletin of the Section of Logic*, 30(4):194–204, 2001.
57. Mariusz Urbański. Synthetic tableaux and erotetic search scenarios: Extension and extraction. *Logique et Analyse*, 173-174-175:69–91, 2001.
58. Mariusz Urbański. *Rozumowania abdukcyjne. Modele i procedury.* Wydawnictwo Naukowe UAM, Poznań, 2009.
59. Mariusz Urbański and Andrzej Wiśniewski. On search for law-like statements as abductive hypotheses by socratic transformations. In C. Baskent, editor, *Perspectives on Interrogative Models of Inquiry. Developments in Inquiry and Questions*, pages 111–127. Springer, 2016.
60. Maarten H. Van Emden and Robert A. Kowalski. The semantics of predicate logic as a programming language. *Journal of the ACM (JACM)*, 23(4):733–742, 1976.
61. Paul J Werbos. Backpropagation through time: what it does and how to do it. *Proceedings of the IEEE*, 78(10):1550–1560, 1990.
62. Andrzej Wiśniewski. Socratic proofs. *Journal of Philosophical Logic*, 33(3):299–326, 2004.