Technische Universität Dresden
Institute of Artificial Intelligence

Bachelor Thesis

Extracting Propositional Logic Programs From Neural Networks: A Decompositional Approach

Valentin Mayer-Eichberger

January 24, 2006

Supervised by Prof. Steffen Hölldobler
and M.Sc. Sebastian Bader

Task and General Information

student:

name	Valentin Mayer-Eichberger
matr. number	2889037
date and place of birth	October 14th 1982, Tübingen

task: Construction and evaluation of a new decompositional extraction algorithm in the field of neural symbolic integration.

Abstract

Combining artificial neural networks and logic programming for machine learning tasks is the main objective of neural symbolic integration. One important step towards practical applications in this field is the development of techniques for extracting symbolic knowledge from neural networks.

In this thesis a new extraction method is proposed and thoroughly investigated. It translates the class of feedforward networks with binary threshold functions into propositional logic programs by means of a decompositional approach.

Contents

1 Introduction and Motivation

Artificial neural networks are a well known concept of machine learning and widely used in practical applications. They have shown to perform very good at learning and generalizing from examples and there are established algorithms for such tasks. But the trained knowledge which is represented within the network, is rather hidden and hard to understand. The network produces an answer for a given query, but a reason for particular conclusions is not supplied. Its inside is often viewed as a "black box". Therefore it is important to transform the stored knowledge for further investigation and application into a more human readable version. Logic programs, contrary to neural approaches, are understandable and explain their process of inference. However they are not good at learning tasks. Why not combine and use the advantages of both to overcome their shortcomings? The combination of neural networks and logic programs is the main objective of neural symbolic integration and currently there is a lot of interest, and research in this field ([GBG02], [BH05]).

Imagine a learning task with some background knowledge expressed in form of a logic program and additionally some real world examples, which are not captured by the program: how do we enhance the program by those examples? According to the neural symbolic integration paradigm, this program is embedded into a neural network and improved by training. The knowledge would increase in quality and reliability. In order to realize the refined knowledge in the former setting, the new program has to be extracted from the network. This task is the subject of this thesis.

Challenges of an extraction lie in its time complexity and the comprehensibility of the refined rules ([Dar00]). There are two basic approaches to extraction, the pedagogical and the decompositional one. The first one views a network as a black box and queries all possible inputs. Although there are improvements, the general problem of the global approaches is their complexity. Decompositional methods have a better performance, but unfortunately often lose soundness or completeness.

1.1 Previous Work

For a general introductory survey of neural symbolic integration I refer to [BH05] and recommend [ADT95] and [Dar00] for the special task of rule extraction.

In [HK94] a bridge between propositional logic programs and neural networks is established by a transformation algorithm. The algorithm designs a three layered feed forward net with binary threshold functions and adjusts the weights, so that the network computes the semantics of the program. Moreover, for every feed forward neural network a logic program can be generated such that the associated semantic operator has the same input-output behaviour.

The reduction of propositional programs to several minimality criteria is analyzed in [Leh05], especially if they are provided by a pedagogical extraction of a neural network. On the one hand it is shown that in the case of positive networks one can compute a unique minimal propositional program. On the other hand, in the normal case a unique minimal logic program does not necessarily exist. I will reuse two examples stated in this work and refer to the minimality algorithms.

The extraction algorithm "MofN" is presented in [ToSh93] for the "Knowledge Based Artificial Neural Network" (KBANN), an example for a neural symbolic system. This decompositional method is concerned about drawing rules of the form "if M literals of the set of N literals are true then follows..." from one neuron.

In [GBG02] the method of [HK94] is extended by relating proposition logic programs and networks of sigmoidal neurons. They design a "Connectionist Inductive Learning and Logic Programming System", short CIL^2P, which realizes the ideas of neural-symbolic integration and apply it to real world problems. Moreover, they develop an extraction algorithm upon networks with sigmoidal neurons. In this extraction, subset orderings of the global search space are taken into consideration and some pruning rules are specified. The algorithm is sound and complete. However, the results are restricted to the structure of weights in the network and loses those properties if applied to arbitrary networks.

1.2 Goals

I want to design an algorithm that accomplishes a complete and sound extraction of a feed forward neural network with binary threshold functions. The resulting program should be as small as possible, i.e. reduced with respect to subsumption and unsatisfiable clauses. Contrary to pedagogical extraction methods, where the reduction steps are applied towards the "end" of the extraction, I will use a decompositional approach. That means, the network is broken up into subnetworks and an extraction is done on those substructures. My leading idea is to apply reduction steps already on those substructures. To do so, I state an algorithm that extracts the knowledge of a single *binary* neuron in terms of its incoming neurons. I will provide a method to reassemble and put those extracted parts together. To preserve completeness and soundness it requires the construction of two connectives and their algebraic properties. In the developed algebra the whole composition can be formulated and the soundness be proven.

1.3 Structure of the Thesis

In section 2 propositional logic programs, artificial neural networks and some algebraic structures are briefly introduced. The following section 3 is centered around developing an algorithm which shows how to extract symbolic knowledge from one single neuron. This is done by viewing the semantics of a neuron in an abstract way and transforming it into a set. The algebra behind those sets is thereafter examined wrt. equality in logic programs and I will define equivalence relations which capture different levels of reduction. Consequently the main theorem is stated which relates the semantics of logic programs and the equivalence classes. In section 4, I am concerned with the actual extraction of a network. This is followed in 5 by a discussion of my results and finally an outlook on improvements and further work.

2 Preliminaries

I will briefly introduce logic programming and neural networks to the extend it is needed for my thesis. For a more complete and detailed introduction I refer to [Apt97] and

[Roj02] respectively. For a primary on algebraic structures I refer to [LiPi97].

2.1 Propositional Logic Programs

Propositional programs are a subset of propositional logic. The language of propositional logic is build up from a set of propositional variables (or atoms) and the known logic connectives. A literal denotes an atom or its negation. A propositional logic program P consists of a set of normal clauses. A normal clause is an implication with a single atom in the head and a finite set of literals in the body. i.e. of the structure $H \leftarrow L_1, L_2 \ldots L_n$, where H is an atom and L_i are literals.

Throughout this thesis I will only talk about propositional programs, therefore the term "program" is sometimes used as a synonym.

Example 2.1 *Let p, q, r be atoms. As listed in Figure 1 three logic programs of different size are presented.*

P_1 :	P_2 :	P_3 :
$p \leftarrow \neg p, r$	$p \leftarrow \neg p, r$	$p \leftarrow \neg p, r$
$p \leftarrow q, \neg r$	$p \leftarrow q, \neg r$	$p \leftarrow q, \neg r$
$p \leftarrow p, \neg q$	$p \leftarrow p, \neg r$	$p \leftarrow p, \neg r$
	$p \leftarrow \neg q, r$	$p \leftarrow \neg q, r$
		$p \leftarrow p, \neg q, r$
		$p \leftarrow p, \neg q, \neg r$
		$p \leftarrow p, \neg p$

Figure 1: Logic programs P_1, P_2 and P_3

The set of atoms occurring in the clauses of a program P is called a Herbrand base B_P. An interpretation assigns every atom and formula a truth value, and a Herbrand interpretation I_B can be seen as as a subset of B_P. The semantic operator T_P maps interpretations to interpretations by

$$\begin{aligned} T_P(I) &= \{q \mid q \leftarrow p_1, \ldots, p_n, \neg r_1, \ldots, r_m \in P, n, m \geq 0, \\ &\quad \{p_1, \ldots, p_n\} \subseteq I, \{r_1, \ldots, r_m\} \cap I = \emptyset\} \end{aligned}$$

T_P is also called the *immediate consequence operator* of program P and is a convenient tool for capturing the semantics of a logic program. Mostly it is possible to calculate a model (an interpretation to true) of a program by using the least fixpoint of T_P.

Example 2.2 *The T operator of program P_1:*

$$\begin{aligned} T_{P_1}(\emptyset) &= \emptyset \\ T_{P_1}(\{p\}) &= \{p\} \\ T_{P_1}(\{r\}) &= \{p\} \\ T_{P_1}(\{q\}) &= \{p\} \\ T_{P_1}(\{p,q\}) &= \{p\} \\ T_{P_1}(\{p,r\}) &= \{p\} \\ T_{P_1}(\{q,r\}) &= \{p\} \\ T_{P_1}(\{p,q,r\}) &= \emptyset \end{aligned}$$

Note, that $T_{P_1}(I) = T_{P_2}(I) = T_{P_3}(I)$ for all Interpretations I and thus have the same semantics. Throughout this paper I refer to those examples as P_1, P_2 and P_3.

2.2 Artificial Neural Networks

Throughout this thesis, I will only consider neural networks consisting of a weighted directed acyclic graph of neurons with binary threshold functions. The consequences of this restriction are reviewed in the discussion of my results. The theory of training and designing neural networks is not part of my work and will therefore not be explained here, an introduction for this is [Roj02].

Definition 2.3 *A neuron is a single unit in a neural network and is a tuple (w, θ, Φ, Ψ) defined as:*

- *$w : I \to \mathbb{R}$ is called a weight function and gives every incoming edge a weight. The set I contains all neurons that have a direct edge to this neuron. If the incoming neurons are ordered, w can be seen as a weight vector.*
- *$\vec{i} = \{i_1, i_2, \ldots, i_n\} \in \mathbb{R}^n$ is called the input vector, i.e. the output of the nodes, that have a direct edge this neuron. In my case neurons have only output of $+1$ or -1.*
- *$\theta \in \mathbb{R}$ is the threshold.*
- *$\Phi : \mathbb{R}^n \to \mathbb{R}$ is the activation function. It calculates the activation of the neuron by taking the sum of the weighted input neurons, in my case:*

$$\Phi(\vec{i}) = \vec{w} \cdot \vec{i}$$

- *$\Psi : \mathbb{R} \to \mathbb{R}$ is the output function, also called a binary threshold function:*

$$\Psi(x) = \begin{cases} 1 & x \geq 0 \\ -1 & x < 0 \end{cases}$$

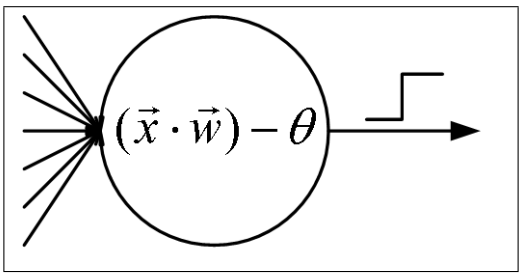


Figure 2: A schematic view of a neuron with a binary threshold function

Since I will only consider neurons that have the same kind of activation and output function, Φ and Ψ are not explicitly stated. So a neuron (as in Figure 2) computes the weighted activation of its connected input, subtracts the threshold and applies the output function. This value is then further used to calculated the activations of the other neurons.

Definition 2.4 *An artificial neural network is a tuple (A, E, w) defined as follows:*

- *(A,E) is a directed, acyclic graph, i.e. A is a set of neurons and $E \in A \times A$ the set of edges that connect neurons.*
- *$w : A \times A \to \mathbb{R}$ is a global weight function, assigning a real number to every edge. The weight function v for a single neuron n is given by the global weight function as $v(a) := w(a, n)$*

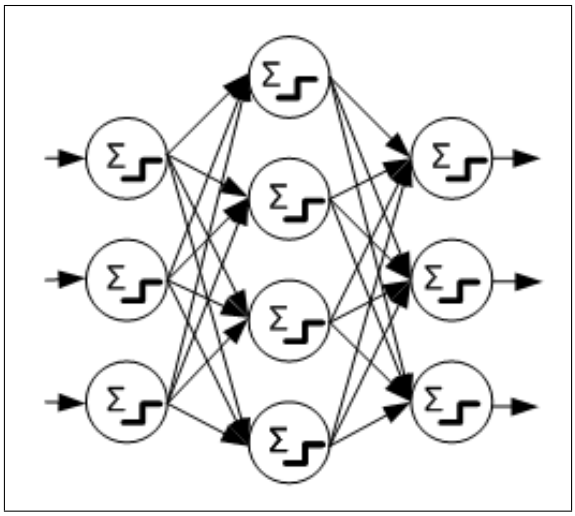


Figure 3: A network with 3 input and output neurons and a single hidden layer of 4 neurons

The computation of the networks follows by propagating the activation of the input layer through the neurons in the hidden layers to the units of the output layer. The neural network as a whole computes a function from $\{+1, -1\}^n$ to $\{+1, -1\}^m$, where n

and m are the number of input and output neurons respectively. However, in this thesis only networks with the same number of input as output neurons are considered. So the network in Figure 3 computes a function from $\{+1, -1\}^3$ to $\{+1, -1\}^3$. A neuron is said to *fire* if it has output of $+1$.

2.3 Algebra

Algebra is the field of mathematical structures. An algebraic structure consists of a set and a collection of operations that have to satisfy certain conditions.

Definition 2.5

A monoid $(M, *, e)$ ia a set M with a binary operation $* : M \times M \to M$ and the following properties:

- *Associativity: for all a, b, c in M, $(a * b) * c = a * (b * c)$*
- *Identity element: there exists an element $e \in M$, such that for all $a \in M$, $a * e = e * a = a$.*

A monoid is commutative if additionally for all $a, b \in M$, $a * b = b * a$.

Definition 2.6

A semiring $(S, +, 0, *, 1)$ is a set with two binary operators satisfying the following conditions:

- *(S,+,0) is a commutative monoid*
- *(S,*,1) is a monoid*
- *Left and right distributivity: For all $a, b, c \in S$, $a * (b + c) = (a * b) + (a * c)$ and $(b + c) * a = (b * a) + (c * a)$.*

A semiring is commutative if additionally the second monoid (S,,1) is commutative.*

Definition 2.7

A Boolean algebra $(A, \vee, 0, \wedge, 1, \neg)$ is a set A, supplied with two binary operations $\wedge$, $\vee$, a unary operation $\neg$ and two elements 0 and 1, such that, for all elements a and b in A, the following axioms hold:

- *associativity and commutativity of $\wedge$ and $\vee$*
- *Left and right distributivity for $\wedge$ and $\vee$.*
- *absorbtion: $a \vee (a \wedge b) = a$ and $a \wedge (a \vee b) = b$*
- *complements: $a \vee \neg a = 1$ and $a \wedge \neg a = 0$*

Definition 2.8

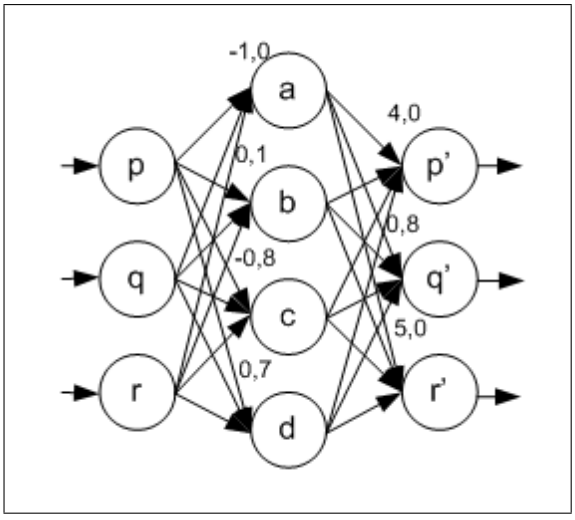
An equivalence relation R on a set A is a binary relation on A that is reflexive, symmetric and transitive, i.e., it holds for all a, b and c in A that

- *(Reflexivity) a R a*
- *(Symmetry) if a R b then b R a*
- *(Transitivity) if a R b and b R c then a R c*

In the analysis of how to compose the divided sub networks into a whole program (Section 3.6) , I use those algebraic structures. In order to put together small parts, two connectives are introduced. Their properties lead me to investigate a semiring. This semiring is then lifted to a boolean algebra by use of an equivalence relation.

2.4 Running Example

Throughout this document I will make use of a running example and show this method of extraction by referring to this example network. Figure 4 shows a drawing of the network and its weight function in form of a table. The number next to the neurons are the thresholds. The example consists of a three layered network with three nodes in the input and output layer and four neurons in the hidden layer.



node	node	weight	node	node	weight
p	a	1,0	a	p'	2,0
p	b	0,5	a	q'	0,0
p	c	0,2	a	r'	-3,3
p	d	-0,5	b	p'	1,0
q	a	1,0	b	q'	-0,9
q	b	0,3	b	r'	2,3
q	c	0,3	c	p'	5,0
q	d	-0,6	c	q'	-1,1
r	a	-2,0	c	r'	-0,1
r	b	-2,1	d	p'	3,0
r	c	0,4	d	q'	0,0
r	d	1,1	d	r'	0,1

Figure 4: Network with 3 input and output neurons and a single hidden layer of 4 neurons and its weight table

2.5 Naive Extraction

How does a pedagogical extraction approach work? The naive method would be to query all possible inputs and derive rules for the program. If the output neuron - referring to a propositional atom p - fires, rules can be deduced by taking the queried input vector implying p. The result for the running example will be referred to as P_4:

$$
\begin{aligned}
&P_4: \\
&p \leftarrow p, \neg q, \neg r \\
&p \leftarrow \neg p, \neg q, r \\
&p \leftarrow \neg p, q, \neg r \\
&p \leftarrow p, q, \neg r \\
&p \leftarrow p, \neg q, r \\
&p \leftarrow \neg p, q, r
\end{aligned}
$$

The extracted program of this method is not very readable and clearly redundant. This algorithm is also not very practical since it is of exponential complexity in the number of input neurons. It would be necessary to check 2^n possible input vectors. The programs P_1 to P_4 all have the same semantic consequence operator. So, there are smaller programs that would be desirable to extract.

3 A Theory of Coalitions

To describe the dynamics of a single neuron, I will develop a notation called *a set of coalitions* which captures all combination that make a neuron fire. Furthermore, I will show that it is sufficient to calculate with a minimal subset of those coalitions and present an algorithm to effectively compute them. Additionally, the counterpart of a coalition is introduced, namely an *opposition set* which stores all combinations that hinder a neuron to fire. The same results also hold for an opposition set.

For the compositional part, the interplay between coalition sets and their connection to logic programs are investigated. The main theorem declares the connection of semantically equal programs and the algebraic structures behind coalition sets and their composition.

3.1 Extended Sets

In this section A will always denote a finite set of symbols $\{a_1, a_2, \ldots, a_n\}$. Every symbol will later refer to a neuron in the network.

Definition 3.1 *$\widetilde{A}$ is called the* extended set *and enriches the set A by a new symbol for every element in A:*

$$\widetilde{A} = A \cup \{\bar{a} \mid a \in A\}$$

As a conventions, $\mathcal{A}$ will denote the powerset of the powerset of A: $\mathcal{A} := \wp(\wp(\widetilde{A}))$. Elements of this set are of great interest and will be examined throughout this thesis. They will be the bridge between logic programs and neural networks. In Section 3.2 the link between programs and $\mathcal{A}$ is shown and in Section 4.1 the connection to networks.

Definition 3.2 *Let $B \subseteq \widetilde{A}$. B is a* base *iff*

- $\forall a \in A$ *either* $\bar{a} \in B$ *or* $a \in B$.
- $\forall a \in A\ \{a, \bar{a}\} \not\subseteq B$

The set $\mathcal{B}$ wrt. A is the set of all possible bases of A. A base captures the possible symbols that can occur, when we want to develop a coalition. They correspond to the sign of the weight of the incoming connection. Bases can be understood as Herbrand interpretations and are used to define the T operator in Section 3.2 .

Definition 3.3 *The function* $\overline{\cdot} : \wp(\widetilde{A}) \to \wp(\widetilde{A})$ *is defined as follows, for all* $Y \subseteq \widetilde{A}$*:*

$$\overline{Y} := \{\bar{a} \mid a \in Y\} \cup \{a \mid \bar{a} \in Y\}$$

Example 3.4 $A = \{p, q, r\}$*, then:*

$$\widetilde{A} = \{p, q, r, \bar{p}, \bar{q}, \bar{r}\}$$

and a base B *could be:*

$$B = \{p, \bar{q}, r\}$$

For this base, the "inverse" $\bar{B}$ *is:*

$$\bar{B} = \{\bar{p}, q, \bar{r}\}$$

On the other hand, $\{p, \bar{p}, r\}$ *would not be a base, because* q *does not occur and both* $p, \bar{p}$ *occur.*

3.2 Power Sets and Logic Programs

The correspondence between power sets and logic programs is crucial for my thesis. It allows me to treat logic programs in an algebraic way. In fact, the definition of the consequence operator T_P can be easily translated into this form.

Let A be the Herbrand base and now looked upon as a set of symbols. It is possible to reformulate a propositional logic programm P as a function from A to $\mathcal{A}$.

$$P(p) = \{\{p_1, \ldots, p_n, \bar{r}_1, \ldots, \bar{r}_n\} \mid p \leftarrow p_1, \ldots, p_n, \neg r_1, \ldots, \neg r_n \in P\} \mid$$

and vice versa. On the one hand, P is a set of clauses (a logic program) and on the other hand it can be viewed as a function $P : A \to \mathcal{A}$. It is only a different notation and therefore the same letter is used.

Example 3.5 *Let* $A = \{p, q, r\}$ *, for example can program* P_1 *be written as a function* $P_1 : A \to \mathcal{A}$ *with* $A = \{p, q, r\}$ *by:* $P_1(p) = \{\{\bar{p}, r\}, \{q, \bar{r}\}, \{p, \bar{q}\}\}$*,* $P_1(q) = P_1(r) = \emptyset$*.*

In the way of expressing logic programs as sets, a Herbrand interpretation, which is a subset of the Herbrand base, would correspond to a base. A Herbrand interpretation I is the same as a base B, when it holds, that $a \in B$ iff $a \in I$ and $\bar{a} \in B$ iff $a \notin I$. Analogously, the semantic operator T can be reformulated.

Definition 3.6 *Let* $P : A \to \mathcal{A}$ *an arbitrary function. The Operator* $T_P : \mathcal{B} \to \mathcal{B}$ *is defined es followed,* $B \in \mathcal{B}$*:*

$$T_P(B) = \{a \in A \mid \exists C \in P(a) \text{ such that } C \subseteq B\} \cup \{\bar{a} \mid a \in A \text{ and } \forall C \in P(a) C \not\subseteq B\}$$

3.3 Coalitions

In Section 3.1 I have established a set $\tilde{A}$ to refer to every node in a network. Likewise, I require a mapping onto the weights in the network which works with the notation of an extended set. Therefore the following definition:

Definition 3.7 *Let $g : A \to \mathbb{R}$ be an arbitrary function. This function is extended to $\tilde{g} : \widetilde{A} \to \mathbb{R}$ by*

$$\tilde{g}(x) := \begin{cases} -g(x) & \text{if } x \text{ is of the form } \bar{y} \\ g(x) & \text{else} \end{cases}$$

Returning to our running example, let g be the weight function between q' and the hidden layer, then $\tilde{g}$ would return a positive value for $\bar{b}$ and $\bar{c}$.

In order to treat all neurons equally, regardless of the signs of the weights, I define a base wrt. the weight function. With this bases and $\tilde{g}$, I am only opposed to units with positive weights. However, the "negated" symbols will later be of great interest.

Definition 3.8 *Let $g : A \to \mathbb{R}$ be an arbitrary function. The base G wrt. g is created by defining*

$$G := \{a \mid g(a) \geq 0\} \cup \{\bar{a} \mid g(a) < 0\}$$

With the following definition I will capture the "least" activation of a neuron, if not all output levels of the input neurons are fixed. The remaining outputs are supposed to be negative wrt. the incentive of the examined neuron. This does not necessarily mean a negative weight and in the extraction it will become clear what is meant by the incentive.

Definition 3.9 *Let $g : A \to \mathbb{R}$ be an arbitrary function and G the base according to g. $|\cdot|_G^g : \wp(G) \to \mathbb{R}$ gives every possible subsets of G a value. Let $C \subseteq G$:*

$$|C|_G^g := \sum_{a \in C} \tilde{g}(a) - \sum_{a \in G \setminus C} \tilde{g}(a)$$

If clear from the context I will omit the g and write $|\cdot|_G$. Also omitting the G is not appropriate, because we are interested in distinguishing between $|\cdot|_G^g$ and $|\cdot|_{\bar{G}}^g$.

Now I can introduce the notation of a coalition, which is a set of symbols for which the value is above a threshold $\theta \in \mathbb{R}$.

Definition 3.10 *Let $g : A \to \mathbb{R}$ be an arbitrary function, G the base wrt. g and $\theta \in \mathbb{R}$.*

- *$C \subseteq G$. If $|C|_G \geq \theta$ then C is called a coalition.*
- *The set $\mathcal{C}_{g,\theta} \subseteq \wp(G)$ contains all possible coalitions.*

$$\mathcal{C}_{g,\theta} = \{C \subseteq G \mid |C|_G \geq \theta\}$$

With the same tools I will define an opposition, a set that states those inputs that have to be fixed in order to keep a neuron from exceeding its threshold. Therefore an opposition chooses the symbols from the "inverted" base.

Definition 3.11 *Let $g : A \to \mathbb{R}$ be an arbitrary function, G the base wrt. g and $\theta \in \mathbb{R}$.*

- *$O \subseteq \bar{G}$. If $|O|_{\bar{G}} < \theta$ then O is called an opposition.*
- *The set $\mathcal{O}_{g,\theta} \subseteq \wp(\bar{G})$ contains all possible opposition.*

$$\mathcal{O}_{g,\theta} = \{O \subseteq \bar{G} \mid |O|_{\bar{G}} < \theta\}$$

Example 3.12 *$A = \{p, q, r\}$, $g(p) = -0,5$,$g(q) = -0,6$, $g(r) = 1,1$. $\theta = 0,7$. Then $G = \{\bar{p}, \bar{q}, r\}$ and for example:*

$$\begin{aligned}
|\{\bar{q}\}|_G &= \tilde{g}(\bar{q}) - (\tilde{g}(\bar{p}) + \tilde{g}(r)) = 0,5 - (-(-0,6) + 1,1) = -1,2 \\
|\{\bar{r}\}|_{\bar{G}} &= \tilde{g}(\bar{r}) - (\tilde{g}(p) + \tilde{g}(q)) = -1,1 - (-0,5 + -0,6) = 0,0 \\
\mathcal{C}_{g,\theta} &= \{\{\bar{p}, r\}, \{\bar{q}, r\}, \{\bar{p}, q, r\}, \{p, \bar{q}, r\}\} \\
\mathcal{O}_{g,\theta} &= \{\{\bar{r}\}, \{p, q\}, \{\bar{r}, p\}, \{\bar{r}, q\}, \{\bar{r}, p, q\}\}
\end{aligned}$$

3.4 Minimal Elements in $\mathcal{A}$

I want to define two functions upon elements in $\mathcal{A}$. The first isolates all elements that are involved in a subset relation with another element. The second excludes all sets that contain a and $\bar{a}$ for some $a \in A$. This minimization corresponds to the reduction steps of logic programs (as in [Leh05]), only here I treat it on the basis of sets.

Definition 3.13 *Let $\mathcal{C} \in \mathcal{A}$. The function $\mu : \mathcal{A} \to \mathcal{A}$ minimizes $\mathcal{C}$ by excluding all elements that have subsets in $\mathcal{C}$.*

$$\mu(\mathcal{C}) = \mathcal{C} \setminus \{C \in \mathcal{C} \mid \exists C' \in \mathcal{C} \ s.t \ .C' \subset C\}$$

Definition 3.14 *Let $\mathcal{C} \in \mathcal{A}$. The function $\eta : \mathcal{A} \to \mathcal{A}$ minimizes $\mathcal{C}$ by excluding all sets that contain a and $\bar{a}$ for some $a \in A$:*

$$\eta(\mathcal{C}) = \mathcal{C} \setminus \{C \in \mathcal{C} \mid \exists a \in A \text{ such that } \{a, \bar{a}\} \subseteq C\}$$

I defined the functions as a set difference, but it can also be stated explicitly. In that case $\eta(\mathcal{C})) = \{C \in \mathcal{C} \mid \nexists a \in A \text{ s.t. } \{a, \bar{a}\} \subseteq C\}$ and $\mu(\mathcal{C}) = \{C \in \mathcal{C} \mid \nexists C' \in \mathcal{C} \text{ s.t. } C' \subset C\}$.

The following proposition shows that those two function can be applied independently and this is important for later proofs.

Proposition 3.15 *For $\mathcal{C} \in \mathcal{A}$, the following holds:*

1. *μ and η are idempotent:*

$$\begin{aligned} \mu(\mu(\mathcal{C})) &= \mu(\mathcal{C}) \\ \eta(\eta(\mathcal{C})) &= \eta(\mathcal{C}) \end{aligned}$$

2.

$$\mu(\eta(\mathcal{C})) = \eta(\mu(\mathcal{C}))$$

3. *Let $\mathcal{C}_{g,\theta}$ be a set of coalitions created by g and θ*

$$\eta(\mathcal{C}_{g,\theta}) = \mathcal{C}_{g,\theta}$$

Proof:

1. Suppose this is not true. Then I would have to find an element in $\mu(\mathcal{C})$, which is not present in $\mu(\mu(\mathcal{C}))$ or converse. But since all element in $\mu(\mathcal{C})$ have no subset relation with each other, this element cannot exist. With the same argumentation, there cannot exist an element in $\mu(\mu(\mathcal{C}))$ that is not contained in $\mu(C)$. This shows idempotency of μ. For η this is done analogously.

2. By applying both definitions we get

$$\mu(\eta(\mathcal{C})) = \{C \in \eta(\mathcal{C}) \mid \nexists C' \in \eta(\mathcal{C}) \text{ s.t. } C' \subset C\}$$

   Evidently it doesn't matter if I first exclude all irrelevant elements (wrt. η) and then test the remaining about subset relation or if I do the exclusion of η after all supersets are separated. Thus

$$\begin{aligned} &\{C \in \eta(\mathcal{C}) \mid \nexists C' \in \eta(\mathcal{C}) \text{ s.t. } C' \subset C\} \\ =&\eta(\{C \in \mathcal{C} \mid \nexists C' \in \mathcal{C} \text{ s.t. } C' \subset C\}) \\ =&\eta(\mu(\mathcal{C}))) \end{aligned}$$

3. Subsets of a base never contain both a and $\bar{a}$, since the definition of a base exclude it. A coalition set contains only subsets of bases, therefore η does not subtract anything. □

If a coalition set is given by a function and a threshold, η has no effect (Proposition 3.15), so I define a minimal coalition to be only minimized by μ. In the following sections I demonstrate that the minimized coalitions are sufficient to capture the knowledge of one neuron. Therefore it is enough to compute only the minimal elements.

Example 3.16 *This shows the minimization of the coalition set and opposition set of the Example 3.12:*

$$\begin{aligned} \mathcal{C}_{g,\theta} &= \{\{\bar{p}, r\}, \{\bar{q}, r\}, \{\bar{p}, q, r\}, \{p, \bar{q}, r\}\} \\ \mu(\mathcal{C}_{g,\theta}) &= \{\{\bar{p}, r\}, \{\bar{q}, r\}\} \\ \mathcal{O}_{g,\theta} &= \{\{\bar{r}\}, \{p, q\}, \{\bar{r}, p\}, \{\bar{r}, q\}, \{\bar{r}, p, q\}\} \\ \mu(\mathcal{O}_{g,\theta}) &= \{\{\bar{r}\}, \{p, q\}\} \end{aligned}$$

3.5 Algorithm to Compute Minimal Coalitions and Opposition

In this section I will introduce a algorithm which computes $\mu(\mathcal{C}_{g,\theta})$ and $\mu(\mathcal{O}_{g,\theta})$ given $g : A \to \mathbb{R}$ and $\theta \in \mathbb{R}$. This algorithm corresponds to an extraction of a neuron into an appropriate notation, namely the minimal coalition and opposition sets.

3.5.1 Search Space

Given a function $g : X \to \mathbb{R}$ and the base G. The search tree is defined as follows:

- Every node is a subset of G
- The root is the empty set $\emptyset$
- The children of a node differ in one element, this element is smaller wrt. g than all elements in the parent.
- The order of the children is according to $\tilde{g}$. descending from right to left. If some elements have the same weight, one takes an arbitrary but fixed order.

Looking from top to bottom, one element is added at every step and the descendants of a node are its supersets. For clearness reasons, every node is referred to by the symbol that separates it from its parent, and not the whole set.

Example 3.17 *In Figure 5 the subnetwork used for the extraction of neuron p' is enlarged. The search tree for the neuron q' would then be build by the base of the incoming neurons $B = \{a, b, c, d\}$ and the weight function, as showed in Figure 6. The left most node in the search tree corresponds to the set $\{a\}$ and lowest right most to $\{a, b, c, d\}$.*

3.5.2 Algorithm

Main objective of the algorithm is to find the minimal elements in a coalition set given by a weight function and a threshold. Those elements correspond to the first nodes found which exceed the threshold, when searched top to bottom. The algorithm bases its pruning on the following two situations that can occur while searching:

- If a minimal node is found, no other minimal nodes can be found below it, because all sub-nodes are super sets.

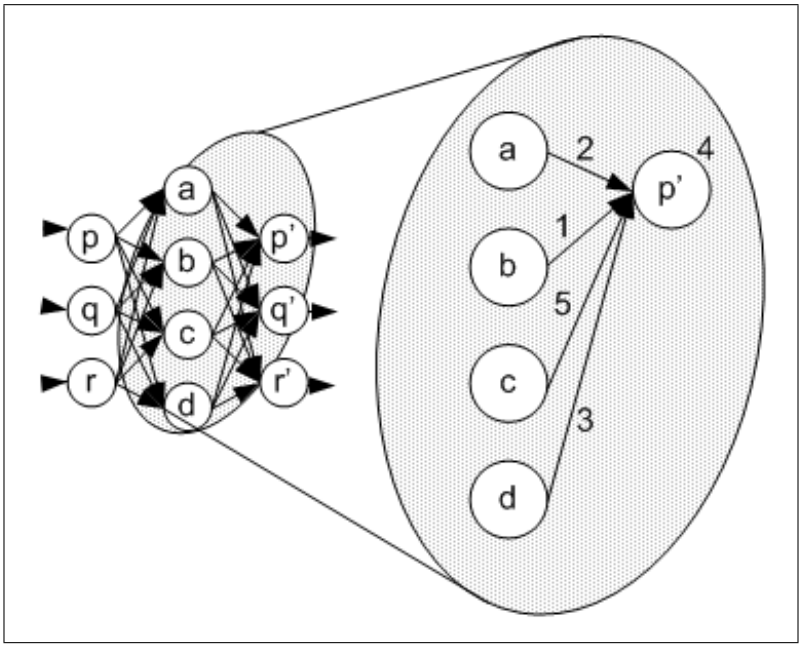


Figure 5: The neuron p' and its input neurons (Example 3.17)

NEURON	WEIGHT
a	2,0
b	1,0
c	5,0
d	3,0

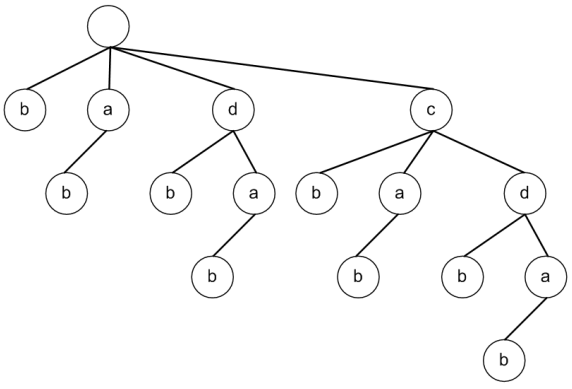


Figure 6: The search tree of neuron p' (Example 3.17)

- If a whole subtree contains no coalitions, then all siblings to the left of the root of this subtree have no either. Additionally, if this root was the right most child of its parent, all siblings of this parent can be cut off as well. This can go on until one node is not the right most or the main root is reached.

An algorithm with deep first search enhanced with those two rules is stated here.

Let $S = \{s_1, s_2, s_3, \dots s_n\}$ be an ordered set of all symbols, with $\tilde{g}(s_1) \leq \tilde{g}(s_2) \leq \dots \leq \tilde{g}(s_n)$, $g : S \to \mathbb{R}$ the weight function and $\theta \in \mathbb{R}$ the threshold. $\mathcal{C}_{g,\theta}$ is the resulting set, initially empty. S and $\mathcal{C}_{g,\theta}$ are globally accessible. First $\emptyset$ is tested and if $|\emptyset|_{g,G}$ is already above the threshold, then the only minimal element is already found: $\mathcal{C}_{g,\theta} = \{\emptyset\}$. Else the function "coalition" with the element s_n is called: $coalition(\{s_n\})$

Algorithm 1: computes a minimal coalition set

function coalition($\{s_{i_1}, s_{i_2}, \dots, s_{i_k}\}$)
  $C := \{s_{i_1}, s_{i_2}, \dots, s_{i_k}\}$
  if($|C|_{g,G} \geq \theta$)
    add C to $\mathcal{C}_{g,\theta}$
    `/* cut off subtree, continue with left sibling */`
    if($i_1 \neq 1$)
      call coalition($\{s_{i_1-1}, s_{i_2}, s_{i_3}, \dots, s_{i_k}\}$)
    elseif($C \neq \{s_1\}$)
      call coalition($\{s_{i_2-1}, s_{i_3}, s_{i_4}, \dots, s_{i_k}\}$)
    endif
  elseif($i_1 \neq 1$)
    `/* if not leaf, continue searching */`
    call coalition($\{s_{i_1-1}, s_{i_1}, s_{i_2}, s_{i_3}, \dots, s_{i_k}\}$)
  else
    `/* if leaf, perform the second cutting step */`
    let l be such that $C = \{s_1, s_2, \dots, s_l, s_{i_l+1}, s_{i_l+2}, \dots, s_{i_k}\}$
    with $l + 1 \neq i_{l+1}$
    if($k > l$)
      call coalition($\{s_l, s_{i_l+2}, \dots, s_{i_k}\}$)
    endif
  endif
endfunction

A detailed discussion of the complexity of this algorithm goes beyond this work. If all weights are of equal size, the resulting set can become exponentially large wrt. the number of elements in the base set, therefore the worst case complexity is exponential. However, the smaller the coalition set gets, the less nodes have to be search and the cutting is relatively effective. Furthermore, the algorithm works fine with negative weights.

Example 3.18 *In Figure 7 the search tree of neuron p' after applying the algorithm is presented. All nodes that have been reached by the algorithm are marked grey and have*

the computed value next to it. The order of calling the procedure "coalition" is stated to the left.

coalition($\{c\}$)
coalition($\{c,d\}$) $\rightarrow$ add
coalition($\{c,a\}$)
coalition($\{c,a,b\}$)$\rightarrow$ add
coalition($\{c,b\}$)
coalition($\{d\}$)
coalition($\{d,a\}$)
coalition($\{d,a,b\}$)

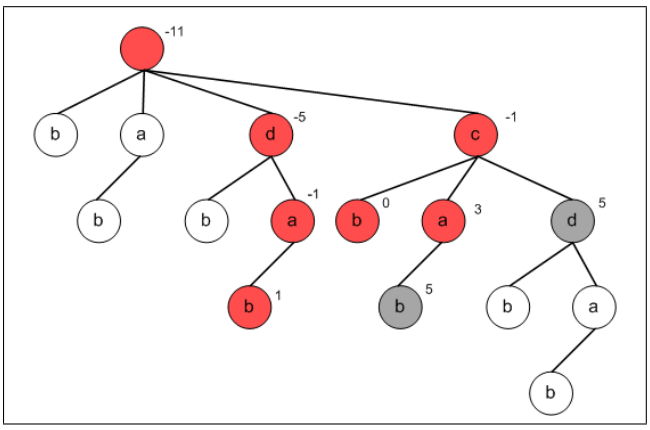


Figure 7: The order of calling the procedure "coalition" and the search tree marked with the nodes that are visited.

Example 3.19 *In this example the extraction of neuron b is visualized in Figure 8. The weights are:* $w(p) = 0,5$, $w(q) = 0,3$, $w(r) = -2,1$ *and the threshold is 0,1. This time the weight for* r *is negative, therefore the base is* $\{p, q, \bar{r}\}$.

coalition($\{\bar{r}\}$)$\rightarrow$ add
coalition($\{p\}$)
coalition($\{q,p\}$)

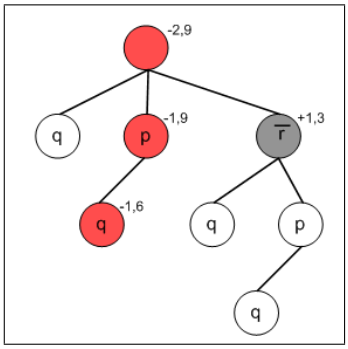


Figure 8: The order of calling the coalition algorithm, visualized by the marked search tree.

Unfortunately the opposition set cannot easily be derived from a given coalition set. Nevertheless the algorithm would be very similar to the one for coalitions. Luckily, only the comparisons of the conditions have to be changed. Instead of testing if $|\mathcal{C}| \geq \theta$, it has to be checked if $|\mathcal{C}| < \theta$. All other steps stay the same, therefore the algorithm will not be stated explicitly here. Note, that elements of oppositions are taken from the "inverted" base $\bar{B}$, because in oppositions, neurons are stated to be not firing.

3.6 Structures for Composing Coalitions

Let me now introduce three algebraic structures. The base set of the first one is $\mathcal{A}$, which represents all syntactical programs. The two last ones build upon equivalence classes of

the first one. Those classes preserve equivalence in a logic programming sematic, whereas the second equivalence relation is stronger than the first one. The second appeared to realize all sets that would be equal under the semantic operator T. The intuition behind the connectives in the algebras is taken from combining coalition sets. In the process of reassembling the decomposed network, "uniting" and "intersecting" coalition and opposition sets is demanded, but unfortunately for intersection this appeared not to be equivalent to the normal set treatment. Therefore I introduce a new operator $\otimes$ and show its algebraic properties.

3.6.1 The Semiring of $\mathcal{A}$

$\mathcal{A}$ is the powerset of the powerset of an extended set $\tilde{A}$.

Definition 3.20 *Let $\mathcal{C}_1, \mathcal{C}_2 \in \mathcal{A}$. The operation $\otimes : \mathcal{A} \times \mathcal{A} \to \mathcal{A}$ is defined as follows:*

$$\mathcal{C}_1 \otimes \mathcal{C}_2 = \{C_1 \cup C_2 \mid C_1 \in \mathcal{C}_1 \text{ and } C_2 \in \mathcal{C}_2\}$$

Lemma 3.21 *The structure $\mathbb{A} = (\mathcal{A}, \cup, \emptyset, \otimes, \{\emptyset\})$ is a commutative semiring.*

Proof: We have to show the following properties:

1. $(\mathcal{A}, \cup, \emptyset)$ is a commutative Monoid.
2. $(\mathcal{A}, \otimes, \{\emptyset\})$ is a commutative Monoid.
3. Between $\otimes$ and $\cup$ the distributivity holds. Let $\mathcal{C}_1, \mathcal{C}_2, \mathcal{C}_3 \in \mathcal{A}$. The following holds:

   $$(\mathcal{C}_1 \otimes \mathcal{C}_2) \cup (\mathcal{C}_1 \otimes \mathcal{C}_3) = \mathcal{C}_1 \otimes (\mathcal{C}_2 \cup \mathcal{C}_3)$$

   and

   $$(\mathcal{C}_1 \cup \mathcal{C}_2) \otimes (\mathcal{C}_1 \cup \mathcal{C}_3) = \mathcal{C}_1 \cup (\mathcal{C}_2 \otimes \mathcal{C}_3)$$

which are shown as follows:

1. Set union $\cup$ is known to be associative and commutative, [LiPi97]. Therefore $(\mathcal{A}, \cup)$ is a commutative semigroup. Since $\emptyset \cup X = X$ for all X, it follows that $\emptyset$ is a unit element.
2. The commutativity of $\otimes$ follows easily from the definition. For associativity:

   $$\begin{aligned}
   (\mathcal{C}_1 \otimes \mathcal{C}_2) \otimes \mathcal{C}_3 &= \{(C_1 \cup C_2 \mid C_1 \in \mathcal{C}_1 \text{ and } C_2 \in \mathcal{C}_2)\} \otimes \mathcal{C}_3 \\
   &= \{C' \cup C_3 \mid C' \in \{(C_1 \cup C_2 \mid C_1 \in \mathcal{C}_1 \text{ and } C_2 \in \mathcal{C}_2)\} \text{ and } C_3 \in \mathcal{C}_3\} \\
   &= \{(C_1 \cup C_2) \cup C_3 \mid C_1 \in \mathcal{C}_1 \text{ and } C_2 \in \mathcal{C}_2 \text{ and } C_3 \in \mathcal{C}_3\} \\
   &= \{C_1 \cup (C_2 \cup C_3) \mid C_1 \in \mathcal{C}_1 \text{ and } C_2 \in \mathcal{C}_2 \text{ and } C_3 \in \mathcal{C}_3\} \\
   &= \mathcal{C}_1 \otimes (\mathcal{C}_2 \otimes \mathcal{C}_3)
   \end{aligned}$$

   It is easy to see that $\{\emptyset\}$ is the unit element.

3.

$$\begin{aligned}(\mathcal{C}_1 \otimes \mathcal{C}_2) \cup (\mathcal{C}_1 \otimes \mathcal{C}_3) &= \{(C_1 \cup C_2 \mid C_1 \in \mathcal{C}_1 \text{ and } C_2 \in \mathcal{C}_2)\} \cup \\ &\quad \{(C_1 \cup C_3 \mid C_1 \in \mathcal{C}_1 \text{ and } C_3 \in \mathcal{C}_3)\} \\ &= \{(C_1 \cup C' \mid C_1 \in \mathcal{C}_1 \text{ and } (C' \in \mathcal{C}_2 \text{ or } C' \in \mathcal{C}_3)\} \\ &= \{(C_1 \cup C' \mid C_1 \in \mathcal{C}_1 \text{ and } C' \in (\mathcal{C}_2 \cup \mathcal{C}_3)\} \\ &= \mathcal{C}_1 \otimes (\mathcal{C}_2 \cup \mathcal{C}_3)\end{aligned}$$

the other distributivity can be shown analogoulsly.

□

Now I will turn back to the two minimization functions that I introduced in Section 3.4 and examine their properties in $\mathbb{A}$.

Proposition 3.22 *For $\mathcal{C}, \mathcal{D} \in \mathbb{A}$ the following holds:*

$$\begin{aligned}1 &: \eta(\mathcal{C} \cup \mathcal{D}) = \eta(\mathcal{C}) \cup \eta(\mathcal{D}) \\ 2 &: \eta(\mathcal{C} \otimes \mathcal{D}) = \eta(\eta(\mathcal{C}) \otimes \eta(\mathcal{D})) \\ 3 &: \mu(\mathcal{C} \cup \mathcal{D}) = \mu(\mu(\mathcal{C}) \cup \mu(\mathcal{D})) \\ 4 &: \mu(\mathcal{C} \otimes \mathcal{D}) = \mu(\mu(\mathcal{C}) \otimes \mu(\mathcal{D}))\end{aligned}$$

Proof:

1. I have to show that all elements that are in $\eta(\mathcal{C} \cup \mathcal{D})$ are also in $\eta(\mathcal{C}) \cup \eta(\mathcal{D})$ and vice versa.
   - For all $E \in \eta(\mathcal{C} \cup \mathcal{D})$ either $E \in \mathcal{C}$ or $E \in \mathcal{D}$. This E would also have to be in $\eta(\mathcal{C})$ or in $\eta(\mathcal{D})$, because it does not contain any symbol together with its counterpart, and is therefore also in $\eta(\mathcal{C}) \cup \eta(\mathcal{D})$.
   - the other direction is shown the same way.
2. I will show that $\eta(\mathcal{C} \otimes \mathcal{D}) \subseteq \eta(\eta(\mathcal{C}) \otimes \eta(\mathcal{D}))$ and $\eta(\eta(\mathcal{C}) \otimes \eta(\mathcal{D})) \subseteq \eta(\mathcal{C} \otimes \mathcal{D})$:
   - Imagine this is not true: Then $\exists E \in \eta(\mathcal{C} \otimes \mathcal{D})$, but $E \notin \eta(\eta(\mathcal{C}) \otimes \eta(\mathcal{D}))$. This E does not contain any illegal occurrence of a symbol ($\nexists a \in As.t.\{a, \bar{a}\} \subseteq E$). For this E there has to be a $C \in \mathcal{C}$ and $D \in \mathcal{D}$ s.t. $C \cup D = E$. From that follows that C and D have no illegal occurrence of any symbol and therefore C is also in $\eta(\mathcal{C})$ and and D in $\eta(\mathcal{D})$. But then E must also be in $\eta(\mathcal{C}) \cup \eta(\mathcal{D})$, which contradicts the assumption.
   - For the other direction analogous steps are done.
3. Here I have to prove that $\mu(\mathcal{C} \cup \mathcal{D}) \subseteq \mu(\mu(\mathcal{C}) \cup \mu(\mathcal{D}))$ and $\mu(\mu(\mathcal{C}) \cup \mu(\mathcal{D})) \subseteq \mu(\mathcal{C} \cup \mathcal{D})$. The first side is shown by proof by contradiction, the other is done similar, but will not be explicitly stated here.

- Suppose this is wrong, then there is an element E in $\mu(\mathcal{C} \cup \mathcal{D})$ which is not in $\mu(\mu(\mathcal{C}) \cup \mu(\mathcal{D}))$. I can assume that this $E \in \mathcal{C}$ and there are no other sets in $\mathcal{C} \cup \mathcal{D}$ that are subsets of E (otherwise E would have been cut by μ). Then E does not have any subsets in $\mathcal{C}$ and is therefore in $\mu(\mathcal{C})$ and in $\mu(\mathcal{C}) \cup \mu(\mathcal{D})$. Because $\mu(\mathcal{C}) \cup \mu(\mathcal{D}) \subseteq (\mathcal{C} \cup \mathcal{D})$ and there where no elements in $\mathcal{C} \cup \mathcal{D}$ that are subsets of E, E must also be in $\mu(\mu(\mathcal{C}) \cup \mu(\mathcal{D}))$, which contradicts the assumption.

4. This is shown likewise to proof 2 and 3.

Now I will make some preparations for the equivalence classes by defining two functions in $\mathbb{A}$, that use the minimization functions η and μ.

$$\begin{aligned} \alpha :& \mathcal{A} \to \mathcal{A} \\ & \mathcal{C} \mapsto \mu(\eta(\mathcal{C}) \end{aligned}$$

and, note that $\mathcal{B}$ are all bases wrt. A:

$$\begin{aligned} \beta :& \mathcal{A} \to \mathcal{A} \\ & \mathcal{C} \mapsto \eta(\mathcal{B} \otimes \mathcal{C}) \end{aligned}$$

Example 3.23 *To get the intuition behind those functions I will use α upon $P_3(p)$ and $P_2(p)$ and β on $P_1(p)$.*

$$\begin{aligned} \alpha(P_3(p)) &= \mu(\eta(\{\{\bar{p}, r\}, \{q, \bar{r}\}, \{p, \bar{r}\}, \{\bar{q}, r\}, \{p, \bar{q}, r\}, \{p, \bar{q}, \bar{r}\}, \{p, \bar{p}\}\})) \\ &= \mu(\{\{\bar{p}, r\}, \{q, \bar{r}\}, \{p, \bar{r}\}, \{\bar{q}, r\}, \{p, \bar{q}, r\}, \{p, \bar{q}, \bar{r}\}\}) \\ &= \{\{\bar{p}, r\}, \{q, \bar{r}\}, \{p, \bar{r}\}, \{\bar{q}, r\}\} \end{aligned}$$

and $P_2(p)$:

$$\alpha(P_2(p)) = P_2(p)$$

so P_2 is minimized already and the two resulting sets are equal, $\alpha(P_2(p)) = \alpha(P_3(p))$.

Note that $\mathcal{B} = \{\{p, q, r\}, \{\bar{p}, q, r\}, \{p, \bar{q}, r\}, \{\bar{p}, \bar{q}, r\}, \{p, q, \bar{r}\}, \{\bar{p}, q, \bar{r}\}, \{p, \bar{q}, \bar{r}\}, \{\bar{p}, \bar{q}, \bar{r}\}\}$. Multiplying with $\mathcal{B}$ and reducing by η is the same as:

$$\eta(\mathcal{B} \otimes \mathcal{C}) = \{B \in \mathcal{B} \mid \exists C \in \mathcal{C} \text{ s.t. } C \subseteq B\}$$

With this simplification the calculation of β is as follows:

$$\begin{aligned} \beta(P_1(p)) &= \eta(\mathcal{B} \otimes \{\{\bar{p}, r\}, \{q, \bar{r}\}, \{p, \bar{q}\}\} \\ &= \{\{\bar{p}, q, r\}, \{p, \bar{q}, r\}, \{\bar{p}, \bar{q}, r\}, \{p, q, \bar{r}\}, \{\bar{p}, q, \bar{r}\}, \{p, \bar{q}, \bar{r}\}\} \end{aligned}$$

and for P_2:

$$\begin{aligned} \beta(P_2(p)) &= \eta(\mathcal{B} \otimes \{\{\bar{p}, r\}, \{q, \bar{r}\}, \{p, \bar{r}\}, \{\bar{q}, r\}\} \\ &= \{\{\bar{p}, q, r\}, \{p, \bar{q}, r\}, \{\bar{p}, \bar{q}, r\}, \{p, q, \bar{r}\}, \{\bar{p}, q, \bar{r}\}, \{p, \bar{q}, \bar{r}\}\} \end{aligned}$$

It seems that applying β is similar to the T operator, since we found out that $\beta(P_1(p)) = \beta(P_2(p))$ and the two programs had also equal semantics.

Proposition 3.24 *Properties of α,β. $\mathcal{C}, \mathcal{D} \in \mathcal{A}$*

$$\begin{aligned}
&1 : \alpha(\mathcal{C} \cup \mathcal{D}) = \alpha(\alpha(\mathcal{C}) \cup \alpha(\mathcal{D})) \\
&2 : \beta(\mathcal{C} \cup \mathcal{D}) = \beta(\mathcal{C}) \cup \beta(\mathcal{D}) \\
&3 : \alpha(\mathcal{C} \otimes \mathcal{D}) = \alpha(\alpha(\mathcal{C}) \otimes \alpha(\mathcal{D})) \\
&4 : \beta(\mathcal{C} \otimes \mathcal{D}) = \beta(\beta(\mathcal{C}) \otimes \beta(\mathcal{D}))
\end{aligned}$$

The proof is mostly technical and deduced from the previous proposition and the observation that $\mathcal{B} = \eta(\mathcal{B}) = \mu(\mathcal{B}) = \eta(\mathcal{B} \otimes \mathcal{B})$:

1.

$$\begin{aligned}
\alpha(\mathcal{C} \cup \mathcal{D}) &= \mu(\eta(\mathcal{C} \cup \mathcal{D}))) \\
&= \mu(\eta(\mu(\eta(\mathcal{C}) \cup \mu(\eta(\mathcal{D})))))) \\
&= \alpha(\alpha(\mathcal{C}) \cup \alpha(\mathcal{D}))
\end{aligned}$$

2.

$$\begin{aligned}
\beta(\mathcal{C} \cup \mathcal{D}) &= \eta(\mathcal{B} \otimes (\mathcal{C} \cup \mathcal{D})) \\
&= \eta((\mathcal{B} \otimes \mathcal{C}) \cup (\mathcal{B} \otimes \mathcal{D})) \\
&= \eta(\mathcal{B} \otimes \mathcal{C}) \cup \eta(\mathcal{B} \otimes \mathcal{D}) \\
&= \beta(\mathcal{C}) \cup \beta(\mathcal{D})
\end{aligned}$$

3. similar to 1.

4.

$$\begin{aligned}
\beta(\mathcal{C} \otimes \mathcal{D}) &= \eta(\mathcal{B} \otimes (\mathcal{C} \otimes \mathcal{D})) \\
&= \eta(\eta(\mathcal{B}) \otimes \eta(\mathcal{C} \otimes \mathcal{D})) \\
&= \eta(\eta(\mathcal{B} \otimes \mathcal{B} \otimes \mathcal{B}) \otimes \eta(\mathcal{C} \otimes \mathcal{D})) \\
&= \eta(\mathcal{B} \otimes \mathcal{B} \otimes \mathcal{B} \otimes \mathcal{C} \otimes \mathcal{D}) \\
&= \eta(\mathcal{B} \otimes (\mathcal{B} \otimes \mathcal{C}) \otimes (\mathcal{B} \otimes \mathcal{D})) \\
&= \eta(\mathcal{B} \otimes \eta(\mathcal{B} \otimes \mathcal{C}) \otimes \eta(\mathcal{B} \otimes \mathcal{D})) \\
&= \beta(\beta(\mathcal{C}) \otimes \beta(\mathcal{D}))
\end{aligned}$$

□

3.6.2 The Semiring $\mathbb{A}_\sim$

Definition 3.25 *I define the following relation $\sim \subseteq \mathcal{A} \times \mathcal{A}$. $\mathcal{C}_1, \mathcal{C}_2 \in \mathcal{A}$:*

$$\mathcal{C}_1 \sim \mathcal{C}_2 \text{ iff } \alpha(\mathcal{C}_1) = \alpha(\mathcal{C}_2)$$

It is easy to see that $\sim$ is an equivalence relation, since the definition is done by an equivalence sign.

Definition 3.26 *Let $[\mathcal{C}]_\sim$,$[\mathcal{D}]_\sim$ be two equivalence classes. The two operators $\tilde{\cup}$ and $\tilde{\otimes}$ are defined as*

$$[\mathcal{C}]_\sim \tilde{\otimes} [\mathcal{D}]_\sim = [\mathcal{C} \otimes \mathcal{D}]_\sim$$

and

$$[\mathcal{C}]_\sim \tilde{\cup} [\mathcal{D}]_\sim = [\mathcal{C} \cup \mathcal{D}]_\sim$$

Proposition 3.27 *$\tilde{\otimes}$ and $\tilde{\cup}$ are well defined.*

Proof:
I have to show that the definition is independent of the chosen representatives, $\forall \mathcal{C}' \in [\mathcal{C}]_\sim$ and $\forall \mathcal{D}' \in [\mathcal{D}]_\sim$ we have to show: $[\mathcal{C}' \cup \mathcal{D}']_\sim = [\mathcal{C} \cup \mathcal{D}]_\sim$ and $[\mathcal{C}' \otimes \mathcal{D}']_\sim = [\mathcal{C} \otimes \mathcal{D}]_\sim$:
First we know that $\mathcal{C}' \in [\mathcal{C}]_\sim$ and $\mathcal{D}' \in [\mathcal{D}]_\sim$.

$$\begin{aligned}
[\mathcal{C}' \cup \mathcal{D}']_\sim &= [\alpha(\mathcal{C}' \cup \mathcal{D}')]_\sim \\
&= [\alpha(\alpha(C') \cup \alpha(\mathcal{D}'))]_\sim \\
&= [\alpha(\alpha(C) \cup \alpha(\mathcal{D}))]_\sim \\
&= [(\mathcal{C} \cup \mathcal{D})]_\sim
\end{aligned}$$

The proof for $\otimes$ is similar $\square$

By showing this, $\mathbb{A}_\sim = (\mathcal{A}/\sim, \tilde{\cup}, [\emptyset]_\sim, \tilde{\otimes}, [\{\emptyset\}]_\sim)$ inherits all properties of $\mathbb{A}$. An equivalence relation as $\sim$ which is compatible with some algebraic operators is also called a congruence relation [LiPi97].

This structure is useful for several reasons: First of all, every equivalence relation $[\mathcal{C}]_\sim$ has a unique computable representant $\mu(\eta(\mathcal{C})$ and this representant is a minimal element. Secondly expressions in $\mathbb{A}_\sim$ can be simplified by factoring out and freely applying μ and η.

3.6.3 The Boolean Algebra $\mathbb{A}_\cong$

Definition 3.28 *The relation $\cong \subseteq \mathcal{A} \times \mathcal{A}$ is defined as follows, $\mathcal{C}, \mathcal{D} \in \mathcal{A}$:*

$$\mathcal{C} \cong \mathcal{D} \text{ iff } \beta(\mathcal{C}) = \beta(\mathcal{D})$$

Since $\cong$ is defined by a function, it is an equivalence relation.

Definition 3.29 *Let $[\mathcal{C}]_\cong$,$[\mathcal{D}]_\cong$ be two equivalence classes. The two operators $\hat{\cup}$ and $\hat{\otimes}$ are defined as*

$$[\mathcal{C}]_\cong \hat{\otimes} [\mathcal{D}]_\cong = [\mathcal{C} \otimes \mathcal{D}]_\cong$$

And

$$[\mathcal{C}]_\cong \hat{\cup} [\mathcal{D}]_\cong = [\mathcal{C} \cup \mathcal{D}]_\cong$$

Proposition 3.30 *The Definition of* $\hat{\otimes}$ *and* $\hat{\cup}$ *is well defined.*

This is shown analogously to $\tilde{\otimes}$ and $\tilde{\cup}$, see proof of Proposition 3.27.

Again, I define $\mathbb{A}_{\cong} = (\mathcal{A}/_{\cong}, \hat{\cup}, [\emptyset]_{\cong}, \hat{\otimes}, [\{\emptyset\}]_{\cong})$ and $\mathbb{A}_{\cong}$ inherits all properties from the former semiring. The isomorphism between the boolean connectives and set operations are a well known result, [LiPi97]. A similar connection could not be found for the first two structures. But for $\mathbb{A}_{\cong}$ the correspondence is established, and in the new factor algebra, union and "intersection" correspond to "and" and "or".

Lemma 3.31 $\mathbb{A}_{\cong}$ *is a Boolean Algebra.*

Proof Sketch:
I will omit the $\hat{}$ above the connectives and the $\cong$ at the equivalence classes. Most of the properties of a Boolean algebra follow from what has been shown before:

- $\vee = \cup$ and $\wedge = \otimes$
- distributivity holds
- $0 = [\emptyset]$ and $1 = [\{\emptyset\}]$
- 1. absorbtion: $x \vee (x \wedge y) = x$

$$\begin{aligned}
[\mathcal{C}] \otimes ([\mathcal{C}] \cup [\mathcal{D}]) &= [\mathcal{C} \otimes \mathcal{B}] \otimes ([\mathcal{C} \otimes \mathcal{B}] \cup [\mathcal{D} \otimes \mathcal{B}]) \\
&= ([\mathcal{C} \otimes \mathcal{B}] \otimes [\mathcal{C} \otimes \mathcal{B}]) \cup ([\mathcal{C} \otimes \mathcal{B}] \otimes [\mathcal{D} \otimes \mathcal{B}]) \\
&= [\mathcal{C} \otimes \mathcal{B}] \cup [\{B \in \mathcal{B} \mid \exists C \in (\mathcal{C} \cap \mathcal{D}) : C \subseteq B\}] \\
&= [\mathcal{C} \otimes \mathcal{B}] \\
&= [\mathcal{C}]
\end{aligned}$$

- the other absorbtion is shown similar
- $\neg([\mathcal{C}]) = [\mathcal{B} \setminus (\mathcal{C} \otimes \mathcal{B})]$, because

$$\begin{aligned}
[\mathcal{C}] \cup \neg[\mathcal{C}] &= [\mathcal{C}] \cup [\mathcal{B} \setminus (\mathcal{C} \otimes \mathcal{B})] \\
&= [\mathcal{B} \otimes \mathcal{C}] \cup [\mathcal{B} \setminus (\mathcal{C} \otimes \mathcal{B})] \\
&= [\mathcal{B}] \\
&= [\{\emptyset\}]
\end{aligned}$$

and

$$\begin{aligned}
[\mathcal{C}] \otimes \neg[\mathcal{C}] &= [\mathcal{C}] \otimes [\mathcal{B} \setminus (\mathcal{C} \otimes \mathcal{B})] \\
&= [\mathcal{B} \otimes \mathcal{C}] \otimes [\mathcal{B} \setminus (\mathcal{C} \otimes \mathcal{B})] \\
&= [(\mathcal{B} \otimes \mathcal{C}) \otimes (\mathcal{B} \setminus (\mathcal{C} \otimes \mathcal{B}))] \\
&= [\emptyset]
\end{aligned}$$

□

3.6.4 Main Theorem

The equivalence classes collect equal coalition sets, but how correspond $\sim$ and $\cong$ to each other? And to what degree are they related to equivalence of programs? The following theorem answers those questions and gives me the right to calculate with an arbitrary element of an equivalence class without worrying about changing the semantics of the related logic program.

Theorem 1 *Let A be a finite set, $\mathcal{A} = \wp(\wp(\widetilde{A}))$. Let $P, Q : A \to \mathcal{A}$ be arbitrary functions.*

$$\begin{aligned} 1: \quad & \forall a \in A : [P(a)]_\sim = [Q(a)]_\sim \\ \Rightarrow \; & \forall a \in A : [P(a)]_\cong = [Q(a)]_\cong \end{aligned}$$

$$\begin{aligned} 2: \quad & \forall a \in A : [P(a)]_\cong = [Q(a)]_\cong \\ \Leftrightarrow \; & T_P = T_Q \end{aligned}$$

Proof:

1. $\forall a \in A : [P(a)]_\sim = [Q(a)]_\sim \Rightarrow \forall a \in A : [P(a)]_\cong = [Q(a)]_\cong$.
   This is shown easily by the equation $\beta(x) = \beta(\alpha(x))\ \forall x \in \mathcal{A}$. To prove this I use the properties of μ, η and $\mathcal{B}$: $\beta(\alpha(x)) = \eta(\mathcal{B} \otimes \mu(\eta((x))$ and $\mu(\eta(\mathcal{B})) = \mathcal{B}$, therefore $\eta(\mathcal{B} \otimes \mu(\eta((x)))) = \eta(\mu(\eta(\mathcal{B})) \otimes \mu(\eta((x)))) = \eta(\mu(\mathcal{B} \otimes x)) = \eta(\mathcal{B} \otimes x) = \beta(x)$. In addition it is easy to see that $\beta(\beta(x)) = \beta(x)$. Let $a \in A$, from $[P(a)]_\sim = [Q(a)]_\sim$ I conclude that $\alpha(P(a)) = \alpha(Q(a))$:

$$\begin{aligned} [P(a)]_\cong &= [\beta(P(a))]_\cong \\ &= [\beta(\alpha(P(a)))]_\cong \\ &= [\beta(\alpha(Q(a)))]_\cong \\ &= [Q(a)]_\cong \end{aligned}$$

2. and $(\forall a \in A : [P(a)]_\cong = ([Q(a)]_\cong \Leftrightarrow T_P = T_Q)$ is proven by changing the quanti-

fiers. T_P quantifies over the bases and the equivalence classes over the symbols.

$$\begin{aligned}
&\forall a \in A : [P(a)]_{\cong} = [Q(a)]_{\cong} \\
\Leftrightarrow\ &\forall a \in A : \eta(\mathcal{B} \otimes P(a)) = \eta(\mathcal{B} \otimes Q(a)) \\
\Leftrightarrow\ &\forall a \in A : \{B \in \mathcal{B} \mid \exists C \in P(a) \text{ s.t. } C \subseteq B\} = \{B \in \mathcal{B} \mid \exists D \in Q(a) \text{ s.t. } D \subseteq B\} \\
\Leftrightarrow\ &(\forall a \in A)(\forall B \in \mathcal{B}) : ((\exists C \in P(a) \text{ s.t. } C \subseteq B) \Leftrightarrow (\exists C \in Q(a) \text{ s.t. } C \subseteq B)) \\
\Leftrightarrow\ &\forall B \in \mathcal{B} : \{a \in A \mid \exists C \in P(a) \text{ s.t. } C \subseteq B\} = \{a \in A \mid \exists D \in Q(a) \text{ s.t. } D \subseteq B\} \\
\Leftrightarrow\ &\forall B \in \mathcal{B} : T_P(B) = T_Q(B) \\
\Leftrightarrow\ &T_P = T_Q
\end{aligned}$$

□

Example 3.32 *In Example 3.23, I showed that* $[P_2]_{\sim} = [P_3]_{\sim}$ *and* $[P_1]_{\cong} = [P_2]_{\cong}$*, thus* P_1, P_2 *and* P_3 *have the same consequence operator.*

4 The Extraction

In the previous section, I have shown how to compute a coalition set for every neuron and developed an algebraic treatment of those coalition sets. Now, I will show how to use this knowledge to compose the parts according to the net structure into a complete extracted logic program. The idea behind this is to start first with the output neurons and compute their coalitions. Every symbol in those sets refers to a neuron in the hidden layer, so this symbol is "replaced" by its coalition set. If the symbol is negated, the opposition set is taken. This is done recursively until we reach the input layer, there we stop and the resulting set expresses the clause bodies for the output symbols. The proven laws of distributivity and commutativity allow me to factor out symbols and simplify sub expressions.

4.1 Construction

Let y be a neuron, that has input neurons $X = \{x_1, x_2, \ldots, x_n\}$ and a weight function $g(X) \to \mathbb{R}$. The base G is according to g and θ is the threshold. I will use the following conventions:

$$\mathcal{C}_y := \mathcal{C}_{g,\theta}$$
$$\mathcal{O}_{\bar{y}} := \mathcal{O}_{g,\theta}$$

Given a neural network (N, E, w) with the set $I \subseteq N$ of input neurons. This network is interpreted as a function $\text{ANN} : I \to \wp(\wp(\widetilde{I}))$. The following function $out : N \to \wp(\wp(I))$ formulates the computation.

$$out(y) := \begin{cases} \{\{y\}\} & y \in I \\ \bigcup_{C \in \mathcal{C}_y} \left(\bigotimes_{x \in C}(out(x))\right) & y \in N \setminus I \text{ and y is of the form } a \\ \bigcup_{O \in \mathcal{O}_y} \left(\bigotimes_{x \in O}(out(x))\right) & y \in N \setminus I \text{ and y is of the form } \bar{a} \end{cases}$$

This function expresses the computation behind a neuron: A neuron fires if there is a ($\cup$) combination of input neurons that fire at the same time ($\otimes$) such that the activation is above the threshold. All hidden and output neurons depend directly or indirectly to the activations of the neurons of the input layer, therefore we have to resolve every symbol which refers to such a neuron back to it. This is done recursively and stops until the computed set only contain neurons of the input layer. And since the network is feed forward and has no cycles, this process terminates. In the end I have an expression in the semiring $(\wp(\wp(I)), \cup, \emptyset, \otimes, \{\emptyset\})$. This expressions can be simplified by the two minimization functions μ and η any time, since I have proven that all elements in $[out(\cdot)]_\sim$ have the same T operator.

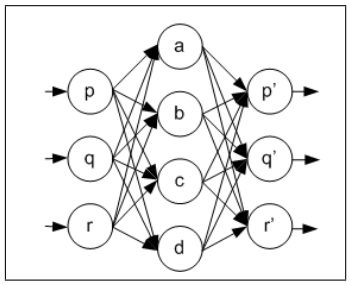
$$[out(y)]_\sim = \begin{cases} [\{\{y\}\}]_\sim & y \in I \\ \tilde{\bigcup}_{C \in \mu(\mathcal{C}_y)} \left(\tilde{\bigotimes}_{x \in C} [(out(x))]_\sim \right) & y \in N \setminus I \text{ and y is of the form } a \\ \tilde{\bigcup}_{O \in \mu(\mathcal{O}_y)} \left(\tilde{\bigotimes}_{x \in O} [(out(x))]_\sim \right) & y \in N \setminus I \text{ and y is of the form } \bar{a} \end{cases}$$

Algorithm 1 from Section 3.5 can be used to compute $\mu(\mathcal{C}_y)$ and $\mu(\mathcal{O}_y)$.

4.2 Soundness and Completeness

The soundness and completeness of my extraction method follow from the Main Theorem 1 stated in Section 3.6.4. The knowledge of an output neuron is captured by the function $out()$ in terms of the input neurons of the network. This function is an expression in the semiring $\mathbb{A}$ and the theorem states, that every element of $[out()]_\sim$ has the same semantics wrt. logic programs. Therefore it is possible to take the minimal coalition and opposition sets for the formulation of $out()$.

4.3 Extraction of the Running Example



x	$\mu(\mathcal{C}_x)$	$\mu(\mathcal{O}_x)$
p'	$\{\{a,b,c\},\{c,d\}\}$	/
q'	$\{\{\bar{b},\bar{c}\}\}$	/
r'	$\{\{\bar{a},b\}\}$	/
a	$\{\{\bar{r}\},\{p,q\}\}$	$\{\{\bar{p},r\},\{\bar{q},r\}\}$
b	$\{\{\bar{r}\}\}$	$\{\{r\}\}$
c	$\{\{p\},\{q\},\{r\}\}$	$\{\{\bar{p},\bar{q},\bar{r}\}\}$
d	$\{\{\bar{p},r\},\{\bar{q},r\}\}$	$\{\{\bar{r}\},\{p,q\}\}$

Figure 9: The coalition and opposition sets for each neuron. For the output neurons the opposition sets are not needed

I will show the computation step by step for the propositional variable p, so I have to calculate the expression for the output neuron p'. In Figure 9, a table with all needed coalition and opposition sets is shown. The extraction is done by applying $out()$ to p'. The application of μ and η are done implicitly, since all elements of the equivalence class

do not change the semantic operator.

$$\begin{aligned} out(p') &= \bigcup_{C \in \mu(\mathcal{C}_{p'})} \left(\bigotimes_{x \in C} (out(x)) \right) \\ &= \bigcup_{C \in \{\{a,b,c\},\{c,d\}\}} \left(\bigotimes_{x \in C} (out(x)) \right) \\ &= \left(\bigotimes_{x \in \{a,b,c\}} (out(x)) \right) \cup \left(\bigotimes_{x \in \{c,d\}} (out(x)) \right) \end{aligned}$$

I show the first part of the union: $\bigotimes_{x \in \{a,b,c\}}(out(x)) = out(a) \otimes out(b) \otimes out(c)$

At this point the coalition sets of a,b,c and d contain only input neurons, therefore I can skip the steps for computing $out()$ for every one of them. I just have to replace them by their coalition set.

$$\begin{aligned} out(a) \otimes out(b) \otimes out(c) &= \{\{\bar{r}\},\{p,q\}\} \otimes \{\{\bar{r}\}\} \otimes \{\{p\},\{q\},\{r\}\} \\ &= \{\{\bar{r}\},\{p,q,\bar{r}\}\} \otimes \{\{p\},\{q\},\{r\}\} \\ &= \{\{p,\bar{r}\},\{q,\bar{r}\},\{p,q,\bar{r}\}\} \\ &= \{\{p,\bar{r}\},\{q,\bar{r}\}\} \end{aligned}$$

The same is done for $\bigotimes_{x \in \{c,d\}} out(x)$:

$$\begin{aligned} out(c) \otimes out(d) &= \{\{p\},\{q\},\{r\}\} \otimes \{\{\bar{p},r\},\{\bar{q},r\}\} \\ &= \{\{\bar{p},r\},\{\bar{q},r\}\} \end{aligned}$$

Finally the two sets are united,

$$\begin{aligned} out(p') &= \left(\bigotimes_{x \in \{a,b,c\}} (out(x)) \right) \cup \left(\bigotimes_{x \in \{c,d\}} (out(x)) \right) \\ &= \{\{p,\bar{r}\},\{q,\bar{r}\}\} \cup \{\{\bar{p},r\},\{\bar{q},r\}\} \\ &= \{\{p,\bar{r}\},\{q,\bar{r}\},\{\bar{p},r\},\{\bar{q},r\}\} \end{aligned}$$

The remaining two output neurons q' and r' are treated here in a more briefly way, since the computation is done analogously to p':

$$\begin{aligned} out(q') &= out(\bar{b}) \otimes out(\bar{c}) \\ &= \{\{r\}\} \otimes \{\{\bar{p},\bar{q},\bar{r}\}\} \\ &= \emptyset \end{aligned}$$

and

$$\begin{aligned} out(r') &= out(\bar{a}) \otimes out(b) \\ &= \{\{\bar{p}, r\}, \{\bar{q}, r\}\} \otimes \{\{\bar{r}\}\} \\ &= \emptyset \end{aligned}$$

So no combination of input neurons will make q' and r' fire, thus no rules can be extracted. The final extracted program is:

$$\begin{aligned} p &\leftarrow p, \neg r \\ p &\leftarrow q, \neg r \\ p &\leftarrow \neg p, r \\ p &\leftarrow \neg q, r \end{aligned}$$

This is equal to program P_2 from the initial example, although it would be more desirable to get the smaller program P_1.

4.4 Complexity

I have pointed out in Section 3.5 that Algorithm 1 for a single neuron is exponential in the worst case. Consequently the overall complexity of the extraction will be exponential as well. But the reassembling of coalition sets expressed by the formula $out()$ is done in polynomial time in the number of nodes in the network and the size of the coalition sets:

Let $m \in \mathbb{N}$ be the number of layers in the network, $k \in \mathbb{N}$ the maximal number of neurons in one layer and $n \in \mathbb{N}$ the maximal number of elements in all coalition and opposition sets that are computed under the extraction. Then the maximal number of applying $\otimes$ and $\cup$ is less than $k \cdot (n \cdot k)^m$. $\otimes$ itself is of quadratic complexity. So in the case of three layers the expenses of the computation of out is polynomial.

5 Conclusion and Further Work

In this work I have presented a decompositional method for extracting propositional logic programs from artificial neural networks. The extraction method works with arbitrary many layers in the network and there is no restriction on the structure of weights.

In Section 3.1 a theoretical background is developed for analyzing small parts of the network. Because negative and positive weights are allowed, an extended set of symbols was created for referring to neurons in two ways, positively and negatively. I showed in Section 3.4 that only a small part of all possible activations of incoming edges have to be taken into consideration. Algorithm 1 in Section 3.5 translates a single neuron given by a weight function and a threshold into the prepared form. In Section 3.6 the composition of extracted subnetworks is theoretically analyzed and in Section 4.1 this is applied on a network. By using the theorem from Section 3.6.4 the soundness and completeness is shown in 4.2.

The programs P_1 to P_4 from the examples above and our running network demonstrate that my extraction is more sufficient compared to the naive pedagogical one. Many redundant steps, as would be performed by a pedagogical extraction, are avoided by using my approach. However, there are cases a network would result in P_1 or P_4. So, the comprehensibility of the extracted rules depends on the given network and its structure.

There are several parts in my extraction algorithm that can be improved or extended.

- In the compositional part a formula is computed and simplified. Based on properties of the algebra, heuristics may be found to improve the simplification.

- Since there might be cases in the computation where only parts of a coalition set is needed, the extraction of a neuron should only be done to the extend it is required. One approach would be that all neuron extractions run in parallel and they "share" their knowledge every time one coalition is found. In this process situations occur in which one neuron does not have to be computed fully or even not at all. I do not know if in general this would decrease the worst case complexity, but it could be seen as an optimization step.

- I could use another notation for extracted rules as in [ToSh93]. They call it (M of N), which means any combination of M elements in the set of literals N imply a literal L. The Notation $\mathcal{A}$ could be enhanced by this notation and thus a reduction of the size of the sets. This means, not all rules have to be written down explicitly if they appear in this pattern and I propose that the extraction of a single neuron could be done in polynomial time in the number of weights. How the operators $\otimes$ and $\cup$ would deal with this extension, I have not tried to develop, but I do not exclude that this can be done.

- As I demonstrated, the extracted program is not necessary of minimal size. The least reduction level that can be assured, is bounded to subsumption and trivially contradictory clauses. The algorithm might do stricter reduction steps, but this occurs by luck and is dependent on the structure of the network. Indented additional reduction steps could be performed at a very early stage, for example after every computation of $\otimes$ and $\cup$. I have not found a practical algorithm to do that. The methods presented in [Leh05] are of high complexity and use a pedagogical or program search approach. In order to develop new methods, perhaps the algebra can be used to prove the soundness of further minimizations.

- The implementation of this theory is an important step and I would like to see how the algorithms work in practise. For this, some research has to go into the translation from sigmoidal to binary activation functions. If this can be done efficiently the extraction algorithm stated here could be tested for practical ability. It can be applied to benchmark problems and be compared to other rule extraction systems.

- With help of the algebra, the structure of networks can be changed without changing the semantics. A formula in $\mathbb{A}$ can easily be transformed into a sum of coalition

sets and thereby be translated into a network structure. For this an algorithm has to be developed, that computes a weight function and a threshold given a coalition set $\mathcal{C}$.

- The most interesting question is about how this approach can be extended to the first order case! How would an algorithm corresponding to my neuron extraction look like? Can the algebra somehow be used to develop a decompositional extraction of networks representing first order programs?

References

[ADT95] Andrew, Diederich and Tickle, *A Survey and Critique of Techniques For Extracting Rules From Trained Artificial Neural Networks*, Knowledge Based Systems 8, 1995.

[Apt97] Apt, *From Logic Programming to Prolog*, Prentice Hall 1997.

[BH05] Bader and Hitzler, *Dimensions of Neural-symbolic Integration - a Structured Survey*, In "We Will Show Them: Essays in Honour of Dov Gabbay", College Publications, 2005.

[Dar00] Darbari, *Rule Extraction from Trained ANN: A Survey*, Technical Report, Institute of Artificial Intelligence, Dept. of Computer Science, Technische Universität Dresden, 2000.

[GBG01] Garcez, Broda and Gabbay, *Symbolic knowledge extraction from trained neural networks: A sound approach*, Artificial Intelligence, 125, 2001.

[GBG02] Garcez, Broda and Gabbay, *Neural-Symbolic Learning Systems - Foundations and Applications*, Springer, Berlin, 2002.

[HK94] Hoelldobler and Kalinke, *Towards a Massively Parallel Computational Model for Logic Programming*, ECCAI 1994.

[LBH05] Lehmann Bader and Hitzler, *Extracting Reduced Logic Programs from Artificial Neural Networks*, *NeSy'05*, 2005.

[Leh05] Lehmann, *Extracting Logic Programs from Artificial Neural Networks*, Grosser Beleg, Technische Universität Dresden, 2005.

[LiPi97] Lidl and Pilz, *Applied Abstract Algebra*, Springer 1997.

[Roj02] Rojas, *Neural Networks: a Systematic Introduction*, Springer 2002.

[TaGh96] Taha and Ghosh, *Three Techniques for Extracting Rules from Feedforward Networks*, Intelligent Engineering Systems Through Artifcial Neural Networks, ASME Press, 1996.

[ToSh93] Towell and Shavlik, *Extracting Refined Rules from Knowledge-Based Neural Networks*, Machine Learning, Volume 13, 1993.

Statement of Academic Honesty

Hereby I declare that this thesis is based on my own thoughts and ideas. No other sources than the ones listed in the bibliography were used.

Valentin Mayer-Eichberger