A PERCEPTRON NETWORK THEOREM PROVER FOR THE PROPOSITIONAL CALCULUS

Marc F.J. Drossaers
Department of Philosophy, Rijksuniversiteit te Utrecht

Logic Group
Preprint Series
No. 46
July 1989

Department of Philosophy
University of Utrecht
Heidelberglaan 2
3584 CS Utrecht
The Netherlands

A Perceptron Network Theorem Prover for the Propositional Calculus

Marc F.J. Drossaers

University of Utrecht, Department of Philosophy
Heidelberglaan 2, 3584 CS Utrecht, The Netherlands

Abstract

In this paper a short introduction to neural networks and a design for a perceptron network theorem prover for the propositional calculus are presented. The theorem prover is a representation of a variant of the semantic tableau method, called the parallel tableau method, by a network of perceptrons. The parallel tableau method is designed to enable determination of the counter-examples of a formula (if any) concurrently.

It is proven that the parallel method is complete, and that the operation of a network of perceptrons that represents the parallel tableau method is as specified in virtue of the parallel tableau method.

The states of all neurons in the network are updated simultaneously. An updated neuron reflects the outcome of a comparison of a weighted sum over the states of the neurons a neuron is connected with, and a threshold value. A network representing a theorem prover that can evaluate the validity of formulas with a maximum length of n atom occurrences, will do so in $2n + 3$ updates of all neurons of such a network.

The size of the network grows exponentially with increasing n. This is thought to be of limited importance, because it does not seem very probable that the validity of very long formulas will have to be evaluated.

Master's Thesis (doctoraalscriptie)
Supervised by:
dr. G.R. Renardel de Lavalette
drs. A.C.C. Coolen
dr. J. H. F. M. Schopman

Utrecht, july 1989

Contents

Introduction

The contents of this paper consists of a short introduction to neural networks and a design for a neural network theorem prover.

The neural network research program, also referred to as Connectionism, may well have started with the invention of the McCulloch-Pitts neuron [McCulloch & Pitts 1943]. A neuron is in fact a binary or, in other kinds of network, a bipolar variable, a variable that is either 1 or 0, or 1 or -1 respectively, the values represent the two extreme states a real neuron can be in. McCulloch and Pitts proved that the behaviour of all networks consisting of McCulloch-Pitts neurons can be described using propositional logic and that for each logical expression satisfying certain conditions there exists a network that behaves as described by that formula.

The next big step was the invention of the perceptron by Frank Rosenblatt [Rosenblatt 1962]. Perceptrons where meant to perform learning tasks. One problem with perceptrons was that perceptrons with hidden units, i.e. neurons that don't represent neither input nor output of the network, could not be used for learning tasks, because it could not be determined which connection weights had to be adapted if the network didn't function well. This problem is known as "the credit assignment problem". Another problem was that perceptrons that did not have hidden units could not perform tasks of significant complexity. Computing XOR was too hard already.

Then a forking of the path took place. Some people continued to work on neural networks as self-organizing (learning) machines. This resulted in the Hopfield model [Hopfield 1982], the PDP models [Rumelhart, McClelland et al. 1986], the Boltzmann Machine [Ackley et al 1985], and others. Other people, amongst whom Minsky [Anderson & Hinton 1981] continued to work on neural networks as computational machines. They worked on the representation of knowledge in networks of connected perceptrons that could be programmed by programming the individual perceptrons.

The people doing research into learning machines developed the so called matrix models. The matrix consists of connection weights, numbers which indicate a measure of the correlation between the states of the connected neurons. In this development also a solution was found for the credit assignment problem.

A drawback of this solution is that a network must use global information. This is thought to be biologically unrealistic, because the adaption of synaptic coefficients (learning) of real neurons is based on local information only, i.e. information a neuron gets from its interactions with other neurons.

Another point is that the current learning procedure for Boltzmann Machines, networks that can peform very complicated tasks, is very inefficient (according to Minsky not much better than exhaustive search for the right connections [Minsky & Papert 1969, 1988]).

The nice things of matrix models of neural networks are that they tend to correct and complete distorted input, and that they are associative memories, i.e. memories in which the contents is retrieved using a part of the stored item as a clue, instead of specifying an adress (as is done in conventional computers). This associative capacity is caused by the distributed representation of items in the network; all neurons con-

tribute to the representation of all items, So the items stored in an neural network are, as it were, piled up in the state space of the neural network; the space defined by all combinations of the values all the neurons of the network.

Neural networks are used to solve problems as well. Problem solving using neural networks with stochastic dynamics, such as the Boltzmann Machine, has another less desired property. Solutions of solved problems are optimal solutions, not the best solutions, or *the* solutions. The reason is that the Boltzmann Machine while evolving obeys soft-constraints, i.e. its operation obeys constraints represented in the network up to a certain measure (due to the stochastic element in the dynamics).

It seems to me that this problem can easily be solved by representing a knowledge representation language, such as predicate logic, in the Boltzmann Machine. Solutions to problems, represented in predicate logic, should then have representations in the Boltzmann Machine that are different to an extend that if the output neurons do represent predicate logic expressions, it can only be *the* solutions. When this is done neural networks can be used for associative reasoning. Associative reasoning is just like associative recall, except that the network does not complete clues to obtain stored items, but clues are posed problems, and completion of the representations means obtaining (*the*) solutions to the problems.

Meanwhile, a lot of research into logic programming is done. The great problem is the huge (potential) searches problem solving using predicate logic leads to, or the lack of fast search methods. This has lead to restrictions in the use of predicate logic in artificial intelligence that I think are unsatisfying.

I think that if predicate logic is implemented in a neural network, satisfying the conditions expressed above, the problem of the extensive searches is solved because of the associative reasoning possibility offered by neural networks. The inherent parallelism of neural networks may also well reduce processing time significantly. It will however be necessary, in order to be sure that solutions are correct, that the operation of the network is provably correct, which requires (given the possibilities) that the structure of a neural network that represents a logic based problem solver, is specified systematically, so that it can be garanteed that the structure represents the task domain correctly, and that the network can be programmed, so that it can be garanteed that it operates as specified. Both possibilities are offered us by perceptron networks, but perceptron networks cannot be used for associative reasoning.

To construct a predicate logic based problem solver, or rather a predicate logic theorem prover, it seems therefore necessary to:

(1) Construct a representation of a theorem prover in a feed forward neural network;
(2) Determine relations between feed forward networks and matrix models of neural networks;
(3) Transform the model of (1) into a matrix model using the relations of (2) to garantee correct operation of the transformed model.

The main goal of this paper is to show that a network of perceptrons can be designed that represents a theorem prover for the propositional calculus, and that the operation of the designed network can be proved to satisfy a certain specification.

The paper starts with a short introduction to neural networks (chapter one), and an introduction to the propositional calculus (chapter two). In chapter three a variant of the semantic tableau method for solving validity problems in propositional calculus, the parallel tableau method, is introduced. This alternative method is proven to be correct (complete). In chapter four the structure and the connection weights of a perceptron network that represents the parallel tableau method are specified, and in chapter five the size and growth of such a network are determined. The growth of the network is exponential (which is fast), if a smaller fragment of the propositional calculus is implemented, still an equivalence class of formulas however, growth can be reduced firmly, but is still exponential. The importance of the fast growth of the network is limited, since it seems improbable that any one would like to test whether a very large logical expression is a theorem. Chapter five also contains suggestions for several extensions of the model; a database and an adaption of the network to accomodate the full propositional calculus.

Acknowledgements

I would like to thank all people who helped and encouraged me, and were willing to spend time and effort in understanding what I could possibly mean. I would like to thank very much Eugénie for her support.

Chapter 1

Neural Networks

This chapter contains an introduction to the Hopfield model of a neural network, as described in [Amit 1986], [Amit et al. 1985] and the Boltzmann Machine [Ackley et al. 1985]. The chapter starts with an informal introduction of some important concepts concerning Hopfield models in paragraph one, paragraph two describes formal aspects, and paragraph three some properties of the Hopfield model.

In the paragraphs four and five, an improvement [Amit et al. 1987] and an extension [Sompolinsky & Kanter 1986] of the Hopfield model is presented. The model is improved by enabling storage of patterns with low activity levels, and correlated patterns, and the Hopfield model is extended with the possibility to evolve through a given sequence of metastable states.

The paragraphs six, seven, and eight contain a description of the Boltzmann Machine.

1.1 Hopfield Models

The Hopfield model was constructed to model parts of the nervous system. It is an answer to the question whether in a network of connected formal neurons, a set of connection weights can be chosen in such a way that an arbitrary selection of random activity patterns can be memorized and retrieved. The Hopfield model is a member of the class of associative memory models.

Formal neurons are threshold automata, they perform a weighted sum over the states of (formal) neurons they are connected with, and go "on" if the sum exceeds the threshold value, and go "off" if it doesn't. Connection weights are neural network analogs of synaptic coefficients. Memorized activity patterns are activity patterns that are stable, self-reproducing, they are attractors under the dynamics of the network, where the dynamics of a neural network is the totality of rules or forces that govern the evolution in time at the microscopic level of a network, the level of neuron states. The level of neural networks is called the macroscopic level. In the model no attention is paid to internal processes of neurons, and the connections are taken to be symmetric, which is not in accordance with biological data.

In a Hopfield model two kinds of evolution are possible. One possibility is that the network evolves swiftly from some initial state to a stable state. This is associative recall of a memorized pattern. Another possibility is that the network wanders through state space a relatively long time, until it accidentally stumbles upon a state that reproduces itself under the dynamics of the network.

The space of all possible states of a neural network, i.e. all combinations of all neural states, can be thought of as an "energy landscape", where an energy level is associated with each possible state of the network. Stable activity patterns are associated with energy minima. If all connection weights are positive, then a Hopfield

model has only two attractors: all neurons on, and all neurons off These minima are also called "the ferromagnetic states", this name is due to the resemblance between the Hopfield model and the theory of magnetic systems. In addition to this, by a mixture of positive and negative connection weights, more (local) energy minima are introduced, which have less energy than the ferromagnetic states, if the threshold value is nil [Hopfield 1982].

All memorized activity patterns can be thought of as local energy minima, and learning a pattern as digging a valley (energy minimum) in the energy landscape. A Hopfield model evolves from an initial state to the nearest local minimum. The nearest local minimum is the system state which correlates more with the instantane state (state of the system at a moment) than any other minimum.

A deeper valley in the energy landscape usually attracts the system more than a shallow one. So if an instantane state correlates equally with two different stable states, the system will evolve toward the lower energy minimum most of the times.

If in a Hopfield model many patterns are embedded, attractors of the system start to differ from the activity patterns the neural network has learnt. This is due to correlations between the learnt patterns. If the system has learnt too many patterns, this correlation destroys the possibility to retrieve any pattern, because the patterns are not well distinguished any more in the network.

1.2 Formal Aspects of the Hopfield Model

$S_i \in \{1, -1\}$ is a bipolar neural variable, $i = 1, \ldots, N$, N is the number of neurons in a network. If $S_i = 1$ neuron i is said to fire at maximum frequence, and if $S_i = -1$ its firing frequency is nil.

J_{ij} is the weight of the connection from S_i to S_j, and since the connections are symmetric, also from S_j to S_i. If $J_{ij} > 0$ then the connection is excitatory, if $S_j = 1$ this neuron tends to activate neuron S_i, if $J_{ij} < 0$ the connection is inhibitory. If $J_{ij} = 0$ S_i and S_j are regarded as not connected. In a Hopfield model all neurons are connected, but connections aren't reflexive.

$$h_i = \sum_j J_{ij} S_j$$

is the post synaptic potential, or total input, on S_i. If the total input exceeds the threshold value, which is assumed to be 0 then $1 \rightarrow S_i$; if $h_i < 0$ then $-1 \rightarrow S_i$, if $h_i = 0$ nothing changes. "$\rightarrow$" denotes value assignment.

Updating states in a Hopield model is a Monte Carlo walk; the neurons are updated in an asynchronous fashion, only one at a time, and the neuron that is updated at a moment is selected by chance. The updating proces doesn't stop.

The evolution mechanism can also be expressed as a stochastic proces in order to model synaptic noise. This was done by Little [little 1974]:

$$P(\overline{S}_i) = \frac{1}{1 + \exp\left[-\beta h_i \overline{S}_i\right]}$$

$\overline{S}_i$ is the state of S_i after updating, $\beta = 1/T$, where T is called noise or temperature. $1/\beta$ is the maximum steepness of the function that relates h_i to the probability that S_i will asume a certain value S_i. The Hopfield model is the limit $\beta \rightarrow \infty$ or $T \rightarrow 0$.

This formulation improves the performance of the model. The standard Hopfield model suffers from "hill- climbing" and spurious states; stable states that are mixtures of learnt patterns. With Little's alternative it doesn't if $0.46 \lesssim T \lesssim 1$.

As mentioned before, a Hopfield model evolves from an initial state to a stable state which is an attractor of the system under the dynamics, while minimizing the energy of the system:

$$E = -1/2 \sum_{ij} J_{ij} S_i S_j = -1/2 \sum_i h_i S_i.$$

$$\Delta E = -\Delta S_i \sum_j J_{ij} S_j = -\Delta S_i h_i$$

where $\Delta S_i = S_i(\text{new}) - S_i(\text{old})$. From this it follows that the energy of the network always decreases if the states of neurons is changed during the updating phase, see also [Coolen 1989]:

$$\text{if } S_i = 1 \text{ and } S_i' = -1 \text{ than } h_i < 0 \text{ so } -(S_i' - S_i)\, h_i < 0, \text{ and}$$

$$\text{if } S_i = -1 \text{ and } S_i' = 1 \text{ than } h_i > 0 \text{ so } -(S_i' - S_i)\, h_i < 0.$$

In Hopfield models with Little dynamics attractors of the system are called minima of the free energy; dynamics incorporating noise make it impossible for the system to reach a minimum completely, the system approaches the minima as close as the noise allows for. The relation between the two kinds of Hopfield model is that an attractor under the Little dynamics is an attractor under the dynamics in a standard Hopfield model, but not necessarily the other way round, as is the case with spurious states.

In Hopfield models learning (changing the J_{ij}) is done using the Hebb-rule:

$$J_{ij} = 1/N \sum_{\mu=1}^{p} \xi_i^\mu \xi_j^\mu$$

Where ξ_j^μ is a bit of a pattern memorized by the Hopfield model. $\mu = 1, \ldots, p$ is an index for activity patterns, $i = 1, \ldots, N$ is an index for the neurons of the network. An attractor under the dynamics is denoted by $\{\xi_i^\mu\}$.

When a Hopfield model has learnt an activity pattern μ, it has learnt the complement pattern of μ as well, i.e. the activity pattern in which $\xi_i = -\xi_i^\mu$. When this learning rule is used, the patterns should be drawn at random random and uncorrelated, where randomness usually boils down to a balance between active neurons and neurons at rest in an activity pattern. More elaborate variants of the Hopfield model do not suffer from this limitation.

1.3 Some Properties of the Hopfield Model

Retrieval quality:

p is the number of attractors, N the number of neurons, if $p << N$, then:

(i) if $T > 1$ the system behaves ergodic; there are no stable activity patterns;

(ii) if $T < 1$ there are 2p attractors, all p patterns and their complements. The retrieval quality is expressed as m, without noise, or m(T) (see below) with noise. $\langle S_i \rangle = \pm \xi_i^\mu \, m(T)$, or: the pattern actually recalled (the attractor of the system under the dynamics) is the learned pattern times the retrieval quality;

(iii) if $T = 0$ retrieval is perfect.

Retrieval quality: $m^\mu = 1/N \sum_i \xi_i^\mu \langle S_i \rangle$; "$\langle\ \rangle$" means "average value after equilibrium has been reached"

m(T) can be determined by computing iteratively:

$$m = \tanh(m/T); \quad \tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$$

m^μ can be used to determine the Hamming Distance d, the distance between an attractor and a learnt pattern. It is expressed as the number of bits in which both patterns differ: $d = N(1 - m^\mu)$.

The formula for m can be used to determine the correlation between an arbitrary microscopic state and an attractor as well; just exchange S_i for $\langle S_i \rangle$.

Storage capacity:

As mentioned before, there is a limit to the number of patterns that can be stored in the network, due to the correlations between the stored patterns. The storage capacity of a Hopfield model is defined as the largest p for which attractors are near the learnt patterns, which is about $0.145 \times N$. If $p > 0.145N$ nothing can be retrieved any more.

Robustness:

Hopfield models are very robust. If eighty percent of the connections is set to nil, (destroyed) the retrieval quality is still 0.9 for $T = 0$.

1.4 Storage of Patterns Representing Various Levels of Activity in Hopfield Models

The Hopfield model is improved to enable storage of biased paterns (patterns with a mean activity level different from 0.5), storage of correlated patterns, and to prevent storage of complement patterns. The model is improved by modifying both the learning rule and the dynamics of the system. The learning rule is modified because of the correlations between learnt patterns which make separate retrieval of highly correlated patterns impossible. The dynamics is modified to fit the levels of activity of

the stored patterns. This modification is necessitated by an increase of the number of spurious states caused by the modified learning rule [Amit et al. 1987].

A biased pattern is a pattern which represents an activity level different from 0.5. The bias of a pattern is determined as:

$$1/2\,(1+a);\ -1 \geq a \geq 1$$

where a is the bias parameter defined as:

$$a_\mu = 1/N \sum_i^N \xi_i^\mu$$

In a standard Hopfield model every neuron has an equal probability to be active or at rest (0.5), so usually the number of active neurons balances the number of neurons at rest. This corresponds to an activity level of 50 %. It is further so that:
(i) if $1/2\,(1+a) = 0.5$ then $a = 0$; in a standard Hopfield model the bias is nil.
(ii) $a_\mu = 1/N \sum_i \xi_i^\mu = 0$ for a pattern $\{\xi_i^\mu\}$ in a standard Hopfield model.
(iii) if the patterns in a Hopfield model are uncorrelated then:

$$\langle\langle \xi^\mu\, \xi^\nu \rangle\rangle = 1/N \sum_i \xi_i^\mu \xi_i^\nu = \delta^{\mu\nu}$$

(iv) if the bias is unequal to nil then the patterns are correlated according to:

$$\langle\langle \xi^\mu\, \xi^\nu \rangle\rangle = \delta^{\mu\nu} + a^2\,(1 - \delta^{\mu\nu})$$; if $\mu \neq \nu$ then this boils down to a^2.

δ is the Kronecker delta, and
$\delta^{\mu\nu} = 1$ if $\mu = \nu$
$= 0$ otherwise.

The modified learning rule overcomes the problem of loss of retrieval quality by correcting the connection weights for the bias parameter. The result is that the learnt patterns, though biased, aren't correlated:

$$\bar{J}_{ij} = 1/N \sum_\mu (\xi_i^\mu - a)\,(\xi_j^\mu - a)$$

Notice that the subtraction is a form of non-locality; the bias parameter is a global kind of information. This is rather unnatural.

The idea behind the new dynamics is that the average activity level is known and fixed, and identical to the mean activity level of the learnt patterns. This restriction is expressed as:

$$1/N \sum_i^N S_i = a$$

only evolutions that satisfy this constraint are possible.

The energy function is:

$$E = -1/2 \sum_{ij} J_{ij} S_i S_j$$

The retrieval quality for this model is:

$$m^{\mu} = 1/N \sum_{i}^{N} \langle S_i \rangle (\xi_i^{\mu} - a)$$

The storage capacity of this model is higher than the storage capacity of a standard Hopfield model, for a = 0.925, the storage capacity is 0.18 N. However, since state space is restricted by a global constraint on the dynamics, less information can be stored in biased networks.

1.5 Temporal Association in Hopfield Models

A neural network "knows" temporal associations if it can autonomously evolve through a sequence or cycle of specific patterns. Hopfield models with only symmetrical connections can't do sequences [Hopfield 1982], they cannot perform transitions between stable states.

One model that does sequences has both symmetrical and asymmetrical connections. The symmetrical connections are used for stabilising patterns, the asymmetrical (temporal) connections are used to induce transitions between successive states.

The dynamics of the model is such that the system rests for a while in a (quasi) stable state before evolving to the next state in the sequence. To make this happen, the influence of the temporal connections is to be delayed. This takes a dynamic memory, denoted by τ, a delay constant, which characterizes the function w(t), the dynamical memory.

The connections for the stable states are defined by:

$$J_{ij}^{1} = 1/N \sum_{\mu=1}^{p} \xi_i^{\mu} \xi_j^{\mu}$$

The connections responsible for the transitions between patters are defined as:

$$J_{ij}^{2} = \lambda/N \sum_{\mu=1}^{p-1} \xi_i^{\mu+1} \xi_j^{\mu}$$

Cycles can be made by defining $\xi_i^{p+1} = \xi_j^{1}$.

λ controls the transiton speed. If λ is small no transtions occur, if λ is large, transitions occur too fast to let patterns be stable for even a short while.

The dynamic evolution of the system is defined as the adaption of spins (S_i) to the local fields (h_i): $S_i(t+1) = \mathrm{sgn}[\, h_i(t) \,]$, with:

$$h_i(t) = h_i^{1}(t) + h_i^{2}(t), \text{ where}$$

$$h_i^{1}(t) = \sum_{j}^{N} J_{ij}^{1} S_j(t), \text{ and}$$

$$h_i^2(t) = \sum_{j}^{N} J_{ij}^2 \bar{S}_j(t);$$

$\bar{S}_j(t)$ is the mean value of the $S_j(t')$, where $t' < t$.

$h_i^2(t)$ gives the mean spin state over the period τ. Its effect will therefore increase if the system state remains constant. If $\lambda > 1$, $h_i^2(t)$ will induce transitions to the next stable pattern $\mu + 1$ in the sequence, but not before the system was in state μ for a period $t_0 \approx \tau$, because the momentum needed for a transition is built up when the system is in a stable state, where $t_0 = 1/2\ (1 - 1/\lambda)$.

The storage capacity of this model is about 0.1N.

1.6 The Boltzmann Machine

Massive parallel networks seem very well suited for constraint satisfaction searches, if suitable search techniques are available. In [Ackley et al.] a learning rule and suitable search method are described.

The network is a Boltzmann Machine, a multilayer neural network. By modification of the connections a generative model of a task domain is built up by learning examples, where generative means that the system produces examples in virtue of the domain it represents with the same probability distribution as the examples it learnt. If, after learning, the network is offered part of an example, it interpretes the partial example by trying to find values for the variables of the internal model that would generate the example.

The Boltzmann Machine is fit for constraint satisfaction tasks if the constraint are soft and many in number. Soft constraints are constraints that need not necessarily be satisfied, but that put a price on violation. To compare with chess: the rules of the game are hard constraints, if not obeyed, no chess is played. The rules to play chess well are soft constraints, if not obeyed, the price of loosing all or most games must be paid. The quality of a solution found by the Boltzmann Machine can be expressed as the total cost of all violated constraints.

The Machine consists of units, which are bipolar variables, connected by bidirectional connections. A connection weight represents a soft pairwise constraint between two hypotheses. A unit represents, when having the value 1, an elementary hypotheses about the domain that is supported by the system at that moment. A positive connection weight means that two hypotheses support one another, a negative connection weight means that two hypotheses should not be accepted both simultaneously.

1.7 Behaviour of the Boltzmann Machine

Every global state of the system can be assigned an energy value. Like the Hopfield model the system will evolve to energy minima. If some units are clamped to

represent a certain input, the system will evolve to the energy minimum that is compatible with the input. The energy of a global state is a measure for the violation of the constraints embodied in the Machine, by the present combination of supported hypotheses. While minimizing its energy, the system evolves to interpretations of the input which increasingly satisfy the constraints of the domain represented by the machine.

The energy is expressed as:

(1) $$E = -1/2 \sum_{ij} w_{ij} S_i S_j$$

and

(2) $$\Delta E_i = 2 S_i \sum_j w_{ij} S_j$$

Where (1) is the energy of a global state, and (2) an indication whether unit j should be turned "on" (if $\Delta E_i > 0$) or not. Just like the Hopfield Model, the Boltzmann Machine suffers from "hill-climbing". The solution is the same, the introduction of noise in the dynamics of the system: if a unit is "off" and the input exceeds a threshold, then it will be put "on", if the unit is "off" and the input is smaller than, or equal to, the threshold, then there is a probability it will be put "on" still. This probability is defined as:

(3) $$p = \frac{1}{(1 + e^{\Delta E_i / T})}$$

(3) is a decision rule which holds for particles with two possible energy states as well. A system of such particles reaches a thermal equilibrium, and something like that holds for a Boltzmann Machine that evolves toward a stable activity pattern. The probability of finding the system in any global state satisfies the Boltzmann distribution:

(4) $$P_\alpha = \frac{e^{-E_\alpha / T}}{\sum_\beta e^{-E_\beta / T}}$$

Where P_α is the probability that stable state α is the current state, E_α is its energy. P_β is the probability that β is the next global state. The probability distribution is independent of the route the system follows toward a stable state. At low temperature the system has a strong preference for evolution toward low levels of free energy, but stabilisation time is long, while at high temperature the system does not have such a strong preference, but stabilization time is short. Techniques as "statistical cooling" and "simulated annealing" combine these properties by starting a search at high temperature where the system stabilizes soon, but search is rude. The temperature is then reduced gradually. This way stabilization time is kept within reasonable bounds, and at every temperature the system stabilizes in a state which

satisfies ever more the constraints of the domain represented by the network. The method fails when solutions are associatd with valleys in the energy landscape that are narrow, deep and isolated.

1.8 The Learning of Boltzmann Machines

The great problem with multilayer networks, as expressed by Minsky and Papert [Minsky & Papert 1969] was the so called credit assignment problem, the problem that it could not be determined which connections should be adapted if the network did not operate properly. This problem has been solved for the Boltzmann Machine.

Since the energy is a linear function of the connection weights, there is a simple relation between the log probabilities of a certain global state and the individual connections [Ackley et al. 1985]:

$$\frac{\delta \ln P_\alpha}{\delta w_{ij}} = -1/T \, S_i^\alpha S_j^\alpha$$

The Boltzmann Machine exist of a number of hidden, and a number of visible units. The visible units are divided into input, and output units. The hidden units can be used to represent domain constraints that underly a number of input vectors, that cannot be represented by pairwise constraints between the visible units.

The structure underlying a domain is modelled by putting a number of environment vectors to the system one by one. This is done by letting the system evolve to a stable state while clamping the visible units. The structure of the environment can be specified by giving a probability distribution over all 2^v possible states of the visible units. The network has a perfect model of the environment if it reaches exactly the same probability distribution over the 2^v states in free evolution. This however requires 2^v hidden units. With a smaller number of hidden units, regularities of the domain can be represented, by which a reasonable similarity of the probability distributions can be reached.

The discrepancy between the environment and its representation by the network is:

$$G = \sum_\alpha P(V_\alpha) \ln \frac{P(V_\alpha)}{P'(V_\alpha)} \tag{6}$$

$P(V_\alpha)$ is the probability of the α-th state of the visible units if their state is determined by the environment. It is the probability that a certain state occurs given a probability distribution of the environment. $P'(V_\alpha)$ is the corresponding probability if the system evolves freely. If $G = 0$ the model is perfect, otherwise $G > 0$.

For gradient descend in G it is necessary to know the partial derivative of G with respect to every connection weight. The probabilities of the global states are determined by their energies (4), which are determined by the connections (1). This leads to:

(7) $$\frac{\delta G}{\delta w_{ij}} = -\frac{1}{T}(P_{ij} - P_{ij}')$$

P_{ij} is the mean probability that units i and j are both "on" if the visible units are clamped. P_{ij}' is the corresponding probability if the system evolves freely. To minimize G it is sufficient to collect both probabilities, and change the connection weights proportionally to the difference of the probabilities. This is called the steepest descend method:

(8) $$\Delta w_{ij} = \varepsilon (P_{ij} - P_{ij}')$$

ε is a measure for the stepsize in the steepest descent method. Although this rule uses only local information, which is biologically realistic, the procedure itself is not biologically realistic, since neurons cannot perform statistics.

Chapter 2

Introduction to the Propositional Calculus, and the Semantic Tableau Method for Solving Validity Problems

This chapter contains definitions, lemmas, and theorems concerning propositional calculus and the semantic tableau method for solving validity problems that will serve as background knowledge for chapter three and four. This chapter is based on introductions by Barth & Krabbe [Barth & Krabbe 1982], Hodges [Hodges 1977], and Klop & Meyer [Klop & Meyer 1987].

2.1 Definition of the Language $\mathbb{T}$

Definition 1 (of the language $\mathbb{T}$)
(a) The symbols of $\mathbb{T}$ are:
1) proposition symbols: A, A', A'', . . ,B , B', B'', . . , Z, Z', Z'', . . ;
2) connectives: $\sim$, &, v, and $\rightarrow$;
3) parentheses (and).
(b) the formulas of $\mathbb{T}$:
1) the proposition symbols are the atomic formulas of $\mathbb{T}$;
2) If φ is a formula of $\mathbb{T}$, then so is $\sim\varphi$;
3) If φ and ψ are formulas of $\mathbb{T}$, then so are $(\varphi \& \psi)$, $(\varphi \vee \psi)$, and $(\varphi \rightarrow \psi)$.

If φ and ψ are formulas of $\mathbb{T}$, then $\sim\varphi$ is the negation of φ; $\varphi\&\psi$ is the conjunction of φ and ψ; $\varphi\vee\psi$ is the disjunction of φ and ψ; and $\varphi\rightarrow\psi$ is the implication of ψ by φ.

Parentheses enclosing negations and atoms, as well as the outmost parentheses will be omitted. The greek letters φ, ψ, χ, sometimes indexed, are used as variables for formulas. They are not symbols of $\mathbb{T}$. ATOM is the set of atoms of $\mathbb{T}$, and FORM is the set of formulas of $\mathbb{T}$. Π, Γ are finite sets of formulas of $\mathbb{T}$.

2.2 Semantics

The semantics of the propositional calculus concern the assignment of truth-values, the values "true" and "false", to atomic and complex formulas. An interpretation is an assignment of truth-values to atoms, and a truth-valuation is an assignment of truth-values to complex formulas.

Definition 2 (of an interpretation)
(i) $\mathbb{t}$ and $\mathbb{f}$ are truth values;
(ii) Let $\Pi \subseteq$ ATOM, an interpretation for Π is a function $\mathbb{I}: \Pi \rightarrow \{\mathbb{t}, \mathbb{f}\}$.

An interpretation $\mathbb{I}$ is called a total interpretation if its domain is ATOM, and a partial interpretation otherwise.

Definition 3 (of a truth valuation)

The truth valuation of formulas of $\mathbb{T}$ induced by an interpretation $\mathbb{I}$ is a function $v_{\mathbb{I}}$: FORM $\rightarrow$ {**t**, **f**, UNDEFINED}:

Sem_{At}	$v_{\mathbb{I}}(\varphi) = \mathbb{I}(\varphi)$ if φ is an atom;
SemT$\rightarrow$	$v_{\mathbb{I}}(\varphi \rightarrow \psi) = \mathbf{t}$ if and only if $v_{\mathbb{I}}(\varphi) = \mathbf{f}$ or $v_{\mathbb{I}}(\psi) = \mathbf{t}$ or both;
SemT&	$v_{\mathbb{I}}(\varphi \,\&\, \psi) = \mathbf{t}$ if and only if $v_{\mathbb{I}}(\varphi) = \mathbf{t}$ and $v_{\mathbb{I}}(\psi) = \mathbf{t}$;
SemTv	$v_{\mathbb{I}}(\varphi \vee \psi) = \mathbf{t}$ if and only if $v_{\mathbb{I}}(\varphi) = \mathbf{t}$ or $v_{\mathbb{I}}(\psi) = \mathbf{t}$ or both;
SemT$\sim$	$v_{\mathbb{I}}(\sim\varphi) = \mathbf{t}$ if and only if $v_{\mathbb{I}}(\varphi) = \mathbf{f}$;
SemF$\rightarrow$	$v_{\mathbb{I}}(\varphi\rightarrow\psi) = \mathbf{f}$ if and only if $v_{\mathbb{I}}(\varphi) = \mathbf{t}$ and $v_{\mathbb{I}}(\psi) = \mathbf{f}$;
SemF&	$v_{\mathbb{I}}(\varphi \,\&\, \psi) = \mathbf{f}$ if and only if $v_{\mathbb{I}}(\varphi) = \mathbf{f}$ or $v_{\mathbb{I}}(\psi) = \mathbf{f}$ or both;
SemFv	$v_{\mathbb{I}}(\varphi \vee \psi) = \mathbf{f}$ if and only if $v_{\mathbb{I}}(\varphi) = \mathbf{f}$ and $v_{\mathbb{I}}(\psi) = \mathbf{f}$;
SemF$\sim$	$v_{\mathbb{I}}(\sim\varphi) = \mathbf{f}$ if and only if $v_{\mathbb{I}}(\varphi) = \mathbf{t}$.

If $\mathbb{I}$ is a total interpretation, only Sem_{At} and the SemT rules will be used.

The symbols: $\sim$, &, v, $\rightarrow$ are operators on {**t**, **f**}, the order represents relative priority. We have: $\sim\mathbf{t} = \mathbf{f}$, $\sim\mathbf{f} = \mathbf{t}$;

$$\mathbf{t}\&\mathbf{t} = \mathbf{t},\ \mathbf{t}\&\mathbf{f} = \mathbf{f}\&\mathbf{t} = \mathbf{f}\&\mathbf{f} = \mathbf{f};$$
$$\mathbf{t}\vee\mathbf{t} = \mathbf{t}\vee\mathbf{f} = \mathbf{f}\vee\mathbf{t} = \mathbf{t},\ \mathbf{f}\vee\mathbf{f} = \mathbf{f};$$
$$\mathbf{t}\rightarrow\mathbf{t} = \mathbf{f}\rightarrow\mathbf{t} = \mathbf{f}\rightarrow\mathbf{f} = \mathbf{t},\ \mathbf{t}\rightarrow\mathbf{f} = \mathbf{f}.$$

An example.

$A\,\&\sim A$ is a formula of $\mathbb{T}$. An interpretation for {A} is $\mathbb{I}(A) = \mathbf{t}$, in which case $v_{\mathbb{I}}(A\,\&\sim A) = \mathbf{f}$. Notice that there is no interpretation for which $v_{\mathbb{I}}(A\,\&\sim A) = \mathbf{t}$. This is why the formula is said to be inconsistent.

Definition 4 (of inconsistency)

(i) φ is inconsistent (notation $\varphi\vdash$) if and only if there is no interpretation $\mathbb{I}$ for which $v_{\mathbb{I}}(\varphi)=\mathbf{t}$;

(ii) Π is inconsistent (notation $\Pi\vdash$) if and only if there is no interpretation $\mathbb{I}$ for which $v_{\mathbb{I}}(\varphi)=\mathbf{t}$ for all $\varphi \in \Pi$.

In a similar fashion we can define the notion that a formula entails another formula, or that a set of formulas Π entails a set of formulas Γ.

Definition 5 (of entailment)

(i) φ entails ψ (notation $\varphi \vdash \psi$) if and only if there is no interpretation $\mathbb{I}$ such that:
$v_{\mathbb{I}}(\varphi) = \mathbf{t}$; and
$v_{\mathbb{I}}(\psi) = \mathbf{f}$;
which is called a counter-example for $\varphi \vdash \psi$;

(ii) Π entails Γ (notation $\Pi \models \Gamma$) if and only if there is no interpretation $\mathbf{I}$ such that:
$v_{\mathbf{I}}(\varphi) = \mathbf{t}$ for all $\varphi \in \Pi$; and
$v_{\mathbf{I}}(\psi) = \mathbf{f}$ for all $\psi \in \Gamma$.
which is called a counter-example for $\Pi \models \Gamma$.

A counter-example is also called a partial counter-example if it is a partial interpretation.

Entailment and inconsistency are related; $\Pi \models \varphi$ if and only if $\Pi, \varphi \models$.

If $\Pi = \emptyset$, then $\emptyset \models \varphi$ is written as $\models \varphi$: φ is a theorem or a tautology. A counter-example for $\models \varphi$ is a counter-example for φ.

Definition 6 (of validity)

φ is valid if and only if φ has no counter-examples; otherwise it is invalid.

Statements like "$\Pi \models \Gamma$" are called sequents. Sequents can be correct or incorrect. Sequents like $\Pi \models \Gamma$ express the entailment of Γ by Π. Sequents are not themselves part of $\mathbf{T}$, because "$\models$" is not a symbol of $\mathbf{T}$.

2.3 The Semantic Tableaux Method

The method of semantic tableaux is an efficient method for deciding on the incorrectness of a sequent, i.e. on the existence of a counter-example for $\Pi \models \Gamma$, and, moreover, if it exists, then the method yields a description of such a counter-example.

In order to solve the problem whether a sequent is correct, reduction rules will be applied to it. These rules reduce sequents to simpler sequents, i.e. sequents with less connectives, while maintaining their correctness, until it is a trivial problem whether a sequent is correct.

There are two reduction rules for each operator: one if the operator appears as the principal operator in a formula ψ that is to be given the value T, i.e. $\psi \in \Pi$, and one if the operator appears as the principal operator in a formula ψ that is to be given the value F, i.e. $\psi \in \Gamma$.

Definition 7 (of reduction rules)

The reduction rules of the semantic tableau method are:

$\sim$L:	$\Pi, \sim\varphi \models \Gamma$	is reduced to	$\Pi \models \Gamma, \varphi$;
$\sim$R:	$\Pi \models \Gamma, \sim\varphi$	is reduced to	$\Pi, \varphi \models \Gamma$;
&L:	$\Pi, \varphi \& \psi \models \Gamma$	is reduced to	$\Pi, \varphi, \psi \models \Gamma$;
&R:	$\Pi \models \Gamma, \varphi \& \psi$	is reduced to	$\Pi \models \Gamma, \varphi$ and $\Pi \models \Gamma, \psi$;
vL:	$\Pi, \varphi \vee \psi \models \Gamma$	is reduced to	$\Pi, \varphi \models \Gamma$ and $\Pi, \psi \models \Gamma$;
vR:	$\Pi \models \Gamma, \varphi \vee \psi$	is reduced to	$\Pi \models \Gamma, \varphi, \psi$;
$\rightarrow$L:	$\Pi, \varphi \rightarrow \psi \models \Gamma$	is reduced to	$\Pi \models \Gamma, \varphi$ and $\Pi, \psi \models \Gamma$;
$\rightarrow$R:	$\Pi \models \Gamma, \varphi \rightarrow \psi$	is reduced to	$\Pi, \varphi \models \Gamma\psi$;
C:	$\Pi, \varphi \models \Gamma, \varphi$	is closed.	

Lemma 1.

For each of the left and right rules: an interpretation is a counter-example for the entailment expressed by the sequent on the left of the rule if and only if it is a counter-example for (at least one of) the entailments expressed by the sequent(s) on the right;

Definition 8 (of a semantic tableau)

(a) A semantic tableau for a sequent $\Pi \vdash \Gamma$ based on the reduction rules $\rightarrow$L, $\rightarrow$R, &L, &R, vL, vR, $\sim$L, $\sim$R, and C, is a tree diagram in which each node is associated with a sequent, such that, for each non-final node, the associated sequent is reducible (by one application of a rule) to the sequent(s) associated with its successor(s). The sequent $\Pi \vdash \Gamma$, the original sequent, is itself associated with the root of the tree.

(b) A semantic tableau is closed if and only if each final node is associated with a closed sequent.

Definition 9 (of an open sequent)

A sequent is open if and only if it is not closed.

Definition 10 (of a simple sequent)

A sequent $\Pi \vdash \Gamma$ is simple if and only if $\Pi \cup \Gamma \subseteq$ ATOM.

Lemma 2.

Let $\Pi \vdash \Gamma$ be a simple and open sequent; $\Pi \vdash \Gamma$ is incorrect since one can always find a counter-example for the entailment expressed by an open and simple sequent.

Theorem 1 (completeness theorem)

The sequent $\Pi \vdash \Gamma$ is correct if and only if a semantic tableau for the sequent $\Pi \vdash \Gamma$ can be constructed and closed.

Definition 11 (of decidability)

(a) A decision procedure for a concept pertaining to sequents of a certain kind is a mechanical procedure by which, for a sequent of the appropriate kind, an answer can be obtained (in a finite number of steps) to the question whether the concept applies to the sequent or not;

(b) A concept is decidable if and only if there exists a decision procedure for it.

Theorem 2 (decidability theorem).

The concept of correctness (as pertaining to the language $\mathbb{T}$) is a decidable concept.

Chapter 3

The Parallel Tableau Method for Solving Validity Problems

In this chapter a variant of the semantic tableau will be introduced which enables determination of all counter-examples of a formula (if any) concurrently. This variant is more suitable for implementation in a neural network, in which updating of neurons may be a parallel proces, than the semantic tableau method, which is essentially a sequential method, since it operates, roughly, by reducing a formula to one or two subformula(s), and repeating this until there are only atoms left.

The attraction of parallelism is that time is exchanged for other computational resources such as computer hardware. This may raise the price of computer hardware significantly, but hardware can be bought and time cannot.

The parallel tableau method is meant to be implemented in a neural network and therefore will be described in two ways: as a mathematical abstraction, comparable to the description of the semantic tableau method, and as an assignment of values to a fixed structure of variables, where the fixed structure of variables is a representation of a neural network.

3.1 Some Restrictions

The competence of the neural network theorem prover will be restricted to formulas of $\mathbf{T}$ with $\sim$, &, and v as the only connectives, and the use of the $\sim$ connective restricted to atoms only. This restriction does not have an essential character and is for development purpose only. The set MFORM of formulas, a subset of which a neural network theorem prover is to handle, is defined below.

Another restriction is that the only sequents that will be taken into account have the form $\vdash \varphi$, statements that a formula is a theorem. The neural network model can easily be extended to contain a set DB of literals, i.e. atoms and negations of atoms, that can serve as a database against which the validity of $\vdash \varphi$ can be decided. In that case, although the analysis of complex formulas is limited to sequents $\vdash \varphi$, the correctness is checked as if it were the sequent DB$\vdash \varphi$.

Definition 12 (of a fragment MFORM of $\mathbf{T}$) (in BNF notation, see appendix II)

⟨form⟩ ::= ⟨lit⟩ | ⟨c-form⟩
⟨lit⟩ ::= ⟨atom⟩ | ~ ⟨atom⟩
⟨c-form⟩ ::= ⟨lit⟩ ⟨con⟩ ⟨lit⟩ |
⟨lit⟩ ⟨con⟩ (⟨c-form⟩) |
(⟨c-form⟩) ⟨con⟩ ⟨lit⟩ |
(⟨c-form⟩) ⟨con⟩ (⟨c-form⟩)
⟨con⟩ ::= & | v;
⟨atom⟩ ::= A | A' | A'' . . . Z | Z' | Z'' . . . .

Examples of formulas of MFORM:

$A \vee \sim A$

$(A \& B) \vee ((C \& D) \vee (\sim B \& \sim D))$

Examples of formulas that are not members of MFORM:

$\sim(A \vee B)$

$A \rightarrow B$

$(A) \vee (B)$

3.2 The Parallel Tableau Method

The parallel tableau method boils down to computing concurrently whether a number of relations hold between atom occurrences that occur in the formula under analysis. There are three relations named by the so called tableau predicates, the "in-disjunction predicate, the "negated" predicate, and the "closed" predicate. They replace the right rules, &R, vR, ~R, and C, of the semantic tableau method that are applicable to our fragment of $\mathbb{T}$.

"$\varphi \leq \psi$" means that φ is a subformula of ψ ($\varphi = \psi$ is not excluded).

Definition 13 (of the "in-disjunction" predicate)

Let $\chi \in$ MFORM, φ, ψ are occurrences of atoms of $\mathbb{T}$; φ and ψ are placed in disjunction in χ (notation $D_\chi(\varphi,\psi)$) if and only if there are χ_1, χ_2 such that:

(i) $\chi_1 \vee \chi_2 \leq \chi$ or $\chi_2 \vee \chi_1 \leq \chi$; and

(ii) $\varphi \leq \chi_1$; and

(iii) $\psi \leq \chi_2$.

If φ and ψ are placed in disjunction in χ we say that $D_\chi(\varphi,\psi)$ holds.

The relation expressed by the "in-disjunction" predicate between occurrences of atoms of $\mathbb{T}$, is symmetric, irreflexive, and not-transitive. Symmetric because $D(\varphi,\psi)$ holds if and only if $D(\psi,\varphi)$ holds; irreflexive: $D(\varphi,\varphi)$ is undefined, because the tableau predicate is defined over occurrences of atoms, not over atoms; not-transitive because $D(\varphi,\psi)$ holds if and only if $D(\psi,\varphi)$ holds, but $D(\varphi,\varphi)$ does not hold due to the irreflexiveness of the relation.

Definition 14 (of a full analysis)

A full analysis of a sequent $\vdash \varphi$ is a semantic tableau for $\vdash \varphi$ in which the closure rule has been applied only to simple sequents.

We need the notion of a full analysis to prove functional equivalence of tableau predicates and reduction rules, as stated in theorems 3, 4, and 5, because one cannot say that in a semantic tableau a certain reduction rule will always be applied to formulas that contain atom occurrences for which certain tableau predica-

tes hold, since a semantic tableau may be closed before some reduction rule has been applied to all formulas to which it could have been applied. This cannot happen in a full analysis, which can therefore be well used to proof functional equivalence of tableau predicates and reduction rules.

Theorem 3 rests on the fact that in a full analysis every disjunction of formulas of MFORM is reduced by vR. This is so, because only atoms can be negated, i.e. disjunctions cannot be negated, so vL cannot be applied.

Theorem 3:

Let $\chi \in$ MFORM, φ and ψ are occurrences of atoms of $\mathbb{T}$; $D_\chi(\varphi,\psi)$ holds if and only if there are χ_1, χ_2 such that $\varphi \leq \chi_1$ and $\psi \leq \chi_2$, and in a full analysis of the sequent $\vdash \chi$, vR has been applied to either $\chi_1 \vee \chi_2$ or $\chi_2 \vee \chi_1$.

Proof: see appendix I.

Corollary:

Let $\chi \in$ MFORM, φ and ψ are occurrences of atoms of $\mathbb{T}$; if there are $\chi_1,\chi_2 \leq \chi$ such that $\varphi \leq \chi_1$, $\psi \leq \chi_2$, and $\chi_1 \neq \chi_2$, then $D_\chi(\varphi,\psi)$ does not hold if and only if in a full analysis of the sequent $\vdash \chi$, &R has been applied to either $\chi_1 \& \chi_2$ or $\chi_2 \& \chi_1$.

The corollary states that under certain conditions failure of the "in-disjunction" predicate to hold is equivalent with application of the &R rule.

Definition 15 (of the "negated" predicate)

Let $\chi \in$ MFORM, φ an occurrence of an atom of $\mathbb{T}$; φ is negated in χ (notation $N_\chi(\varphi)$) if and only if $\sim\varphi \leq \chi$;

If φ is negated in χ we say that $N_\chi(\varphi)$ holds.

Since in our fragment of $\mathbb{T}$ only atoms can be negated, in a full analysis every negation of formulas of MFORM is reduced by $\sim$R.

Theorem 4:

Let $\chi \in$ MFORM, φ is an occurrence of an atom of $\mathbb{T}$; $N_\chi(\varphi)$ holds if and only if in a full analysis of the sequent $\vdash \chi$, $\sim$R has been applied to $\sim\varphi$.

Proof: see appendix I.

Definition 16 (of the "closed" predicate)

Let $\chi \in$ MFORM, φ,ψ are two occurrences of the same atom; the "closed" predicate holds for φ and ψ (notation $C_\chi(\varphi,\psi)$) if and only if:

(i) $\varphi, \psi \leq \chi$; and

(ii) $D_\chi(\varphi,\psi)$ holds; and

(iii) either $N_\chi(\varphi)$ holds or $N_\chi(\psi)$ holds, but not both;

The relation expressed by the "closed" predicate is not-transitive, symmetric, and irreflexive.

Theorem 5:

Let $\chi \in$ MFORM, φ,ψ are two occurrences of the same atom; $C_\chi(\varphi,\psi)$ holds if and only if in a full analysis of the sequent $\vdash \chi$, all simple sequents that contain both φ and ψ have the form $..,\varphi,..\vdash..,\psi,..$ or $..,\psi,..\vdash..,\varphi,...$

Proof: see appendix I.

Intuitively, a pseudo counter-example of a formula χ, as defined below, can be viewed as a pair of sets Π and Γ of atoms of a simple sequent $\Pi \vdash \Gamma$, associated with a final node of a semantic tableau for the sequent $\vdash \chi$. The word "pseudo" is added to indicate that it is not (yet) determined whether an interpretation of $\Pi \cup \Gamma$ is a counter-example for χ.

Definition 17 (of pseudo counter-examples)

Let $\chi \in$ MFORM; an ordered pair of sets of atom occurrences ‹TAT, FAT› is a pseudo-counter example (PCE) of χ if and only if:

(i) for all $\varphi, \psi \in$ TAT U FAT, $\varphi \neq \psi$: $D_\chi(\varphi,\psi)$ holds; and
(ii) TAT U FAT is complete concerning (i), i.e. if (i) holds for TAT U FAT U $\{\varphi\}$ then $\varphi \in$ TAT U FAT; and
(iii) for all $\varphi \in$ TAT: $N_\chi(\varphi)$ holds; and
(iv) for all $\psi \in$ FAT: $N_\chi(\psi)$ does not hold.

The main difference between a parallel tableau, defined below, and a semantic tableau is that a parallel tableau does not have any node that is not either a root or a final node. The difference is caused by the parallelism, i.e. the absence of a sequence of breaking up formulas into their parts, of the parallel tableau method.

Definition 18 (of a parallel tableau)

a parallel tableau (PT) is an ordered pair ‹φ, PCES›, where $\varphi \in$ MFORM and PCES is the set of all PCE's of φ.

An example of a parallel tableau:
a parallel tableau for $((\sim A \vee B) \& \sim B) \vee (D \& E)$ is:

‹$((\sim A \vee B) \& \sim B) \vee (D \& E)$, {‹{A}, {B,D}›, ‹{B}, {D}›, ‹{A}, {B,E}›, ‹{B}, {E}›}›

Definition 19 (of a closed PT)

(a) a PCE ‹TAT, FAT› of $\chi \in$ MFORM is closed if and only if there are $\varphi \in$ TAT and $\psi \in$ FAT such that $C_\chi(\varphi,\psi)$ holds; otherwise the PCE is open;
(b) a PT ‹χ, PCES› is closed if and only if every PCE $\in$ PCES is closed; otherwise the PT is open.

Lemma 3:

Let ‹χ, PCES› be a PT; a sequent $\Pi \vdash \Gamma$ is a simple sequent in a full analysis of the sequent $\vdash \varphi$ if and only if there is a PCE ∈ PCES ‹TAT, FAT› such that Π = TAT and Γ = FAT.

Proof: see appendix I.

Lemma 4:

If ‹φ, PCES› is an open PT, then φ is invalid and there is at least one PCE ∈ PCES ‹TAT, FAT› which is open, and an interpretation of TAT ∪ FAT such that:

$\mathbb{I}(\varphi) = \mathbf{t}$ for all $\psi \in$ TAT; and

$\mathbb{I}(\varphi) = \mathbf{f}$ for all $\psi \in$ FAT,

is a counter-example for φ.

Proof: see appendix I.

Theorem 6 (completeness theorem):

Let φ ∈ MFORM; φ is valid if and only if a PT ‹φ, PCES› is closed.

Proof: see appendix I.

3.3 A fixed Tree Structure

A parallel tableau is represented by a tree structure of variables, which is in turn represented by a network of perceptrons. A neural network is a limited structure, and therefore, so is the tree structure. This also limits the length of the formulas that can be analysed.

Under certain conditions a tree structure represents a parallel tableau by the values the variables of the tree structure have. To put it in another way: the tree structure is a skeleton for parallel tableaux, and a parallel tableau is an instantiation of a tree structure.

In this paragraph the notions of a tree structure and its parts are explained and defined. A definition of a parallel tableau as a representation of a tree structure is also given.

The domain of a variable x (notation D(x)) is the set of values that x can have.

In definition 20.a the structure of a root is defined, while 20.b contains the domains of the various variables that make up a root.

Definition 20 (of a root)

A root is an ordered m-tuple, that consists of:

(a)(i) n literal ensembles, a literal ensemble has the form "$\sim_i \alpha_i$", i = 1, . . , n, where $\sim_i$ is a unary-operator-variable; and α_i is an atom-variable;

(ii) n-2 lpar-variables, $(_i$, immediately left of the first (leftmost) literal ensemble,
n-2 lpar-variables immediately left of the second literal ensemble,
n-i lpar-variables immediately left of the i-th literal ensemble, $3 \leq i \leq n$;

(iii) 0 rpar-variables, $)_i$, immediately right of the first literal ensemble,
1 rpar-variable immediately right of the second literal ensemble,
i-1 rpar-variables immediately right of the i-th literal ensemble, $1 < i < n-1$;

(iv) n-1 binary-operator-variables, $\Box_i$, at the right of literal ensemble i and its rpar-variables;

(b) (i) $D(\alpha_i) = ATOM \cup \{*\}$;

(ii) $D(\sim_i) = \{\sim, *\}$;

(iii) $D(\Box_i) = \{\&, v, *\}$;

(iv) $D((_i) = \{(, *\}$;

(v) $D()_i) = \{), *\}$.

An example of a root for n = 4 is:

$\langle (_1, (_2, \sim_1, \alpha_1, \Box_1, (_3, (_4, \sim_2, \alpha_2,)_2, \Box_2, (_5, \sim_3, \alpha_3,)_4,)_1, \Box_3, \sim_4, \alpha_4,)_5,)_3 \rangle$.

The formula (A&B)v(~A&C) is represented by this root if it has the following values:

* (* A & * * * B) v (~ A * * & * C) *

In definition 21 a final node is defined as an fn, and final nodes is an ordered m-tuple of fn's. Each fn is an ordered n-tuple of tf-pairs. A tf-pair is an ordered pair of variables that can have atoms or "*" as their values.

Definition 21 (of final nodes) (in BNF notation)

```
<final nodes>      ::=  <lhook> <fn> {, <fn>} <rhook>
<fn>               ::=  <lhook> <tf-pair> {, <tf-pair>} <rhook>
<tf-pair>          ::=  <lhook> <atom-variable> , <atom-variable> <rhook>
<atom-variable>    ::=  <atom> | <filler>
<rhook>            ::=  >
<lhook>            ::=  <
<filler>           ::=  *
<atom>             ::=  A | A' | A" . . . . Z | Z' | Z" . . .
```

an example of final nodes:

```
<   <<*,A>,     <<B,*>,     <<C,*>,
     <B,*>,      <*,*>,      <*,A>,
     <*,*>>,     <*,A>>,     <*,*>>   >
```

The letters x, y, and z, usually indexed, will be used as variables for objects in the root and fn's, which names end with "-variable".

Definition 22 (of comb(n))

$$comb(n) = \begin{cases} \text{(i) } \binom{n}{n/2} & \text{if } n \text{ is even; and} \\ \text{(ii) } \binom{n}{(n+1)/2} & \text{if } n \text{ is odd.} \end{cases}$$

Definition 23 (of a tree structure)

A tree structure is an ordered pair ‹root, final nodes› such that if the root contains n atom-variables, then

(i) the tree structure is said to have a capacity of n atoms; and
(ii) final nodes has comb(n) fn's; and
(iii) each fn has n tf-pairs.

The reason why there are comb(n) fn's in final nodes and n tf-pairs in an fn will be explained in the next paragraph.

Definition 24 (of occurrence numbers)

t = ‹root, final nodes›; the occurrence numbers of t are natural numbers assigned to the atom-variables of the root, when enumerating them from the left to the right such that:

(i) the first atom-variable has occurrence number 1; and
(ii) if an atom-variable has occurrence number i then the next atom-variable has occurrence number i+1.

The letter o, usually indexed, will be used as variable for occurrence numbers.

Definition 25 (of a representation)

a representation is a function R: {t | t is a tree structure} $\longrightarrow$ {p | p is a PT};
let t = ‹root, final nodes› be a tree structure of capacity n,
root = $\langle x_1, x_2, x_3, \ldots, x_j \rangle$, and
$fn_i = \langle\langle y_{11}, y_{12}\rangle, \langle y_{21}, y_{22}\rangle, \langle y_{31}, y_{32}\rangle, \ldots, \langle y_{n1}, y_{n2}\rangle\rangle$ $i \in \{1, \ldots, comb(n)\}$;

(i) $R(t) = \langle R(root), R(\text{final nodes})\rangle$;
(ii) $R(root) = \langle R(x_1)\ R(x_2)\ R(x_3) \ldots R(x_j)\rangle$;
(iv) $R(\text{final nodes}) = \langle R(fn_1), R(fn_2), R(fn_3), \ldots, R(fn_{comb(n)})\rangle$;
(v) $R(fn_i) = \langle \{R(y_{11}), R(y_{21}), \ldots, R(y_{n1})\}, \{R(y_{12}), R(y_{22}), \ldots, R(y_{n2})\}\rangle$; and
(vi) $R(z) = \begin{cases} \text{undefined if } z = *; \\ z \text{ otherwise.} \end{cases}$

3.4 Registration of Occurrences in Final Nodes

Registration is the assignment of values to variables in final nodes. If registration is performed according a certain heuristic, it can be shown that all PCE's of a formula, represented by the root of a tree structure are represented by the fn's of the same tree structure after registration. In this way, registration is a search for the PCE's of formulas.

In this paragraph, registration and the heuristic used to search for PCE's are described. It should be noted that the heuristic described below and occurrence numbers are not represented in a network of perceptrons that represents a tree structure as explicitly as they are presented here. The heuristic is embedded in the connection pattern of such a network, and occurrence numbers coincide with indices of atom-variable representing perceptron predicates. Both appear in this chapter as explanatory devices, used to describe the operation of the perceptron network at the level of a tree structure.

The problem to be solved when registering atoms in final nodes is how to get all PCE's of the formula which atoms are being registered. If two atoms are conjuncts, for instance, these occurrences cannot be in the same PCE and therefore not in the same fn. This will be illustrated with two examples of tree structures.

Example 1:
root: ‹*A&*B›

final nodes:

	fn_1	fn_2
‹	‹‹*, A›,	‹‹*, *›,
	‹*, *››,	‹*, B›› ›

Example 2:
root: ‹**Av(*B*&*C)›

final nodes:

	fn_1	fn_2	fn_3
‹	‹ ‹*,A›	‹ ‹*, A›	‹ ‹*, C›
	‹*, B›	‹*, C›	‹*, A›
	‹*, *› ›	‹*, *› ›	‹*, *› › ›

The problem in example 1 is whether "A" or "B" should be registered in some fn. This is a kind of indeterminacy, since there is, for instance, nothing that prevents the registration of "A" in both fn's, so that there is no fn left for "B" to be registered in. A similar point can be made concerning example 2; whether, when registration starts with "A", "A" should be accompanied by "B" or by "C".

The solution for this problem is to use ordered lists of all the occurrence numbers as registration recipes, one ordered list per fn. the order of the ocurrence numbers will express a priority arrangement for the registration of atom occurrences in an fn. The arrangement is that an occurrence of an atom can be registered in some fn if and only if it is placed in disjunction with all occurrences that have higher priority, and can be registered in that fn themselves.

To make this solution work it is necessary that the occurrence numbers in the lists are ordered the right way. Explaining what the right way is also explains the number of fn's in final nodes, and the number of tf-pairs in an fn.

Let #(TAT) denote the number of elements of TAT. In a tree structure of capacity n in which each fn is dedicated to a class of PCE's that have identical unions of TAT and FAT, it is necessary to have:

- $\binom{n}{1}$ fn's for all PCE's with #(TAT U FAT)= 1; and

- $\binom{n}{2}$ fn's for all PCE's with #(TAT U FAT)= 2; and

.

.

.

- $\binom{n}{n} = 1$ fn for all PCE's with #(TAT U FAT)= n.

The total number of fn's can be determined as follows:

$$\sum_{k=0}^{n} \binom{n}{k} = 2^n$$

$\binom{n}{0} = 1$ and denotes an empty PCE which is not relevant, so the total number of fn's for any n is $2^n - 1$.

Let $occurs_i$ be an ordered n-tuple of occurrence numbers of the occurrences fn_i is dedicated to. Suppose that $occurs_i = \langle 1 \rangle$, registration in fn_i wil then come down to assigning the value of α_1 in the root to one of the two atom-variables in fn_i.

Let $occurs_i'$ be $occurs_i$ with the other n-1 occurrence numbers added to $occurs_i$ in such a way that occurrence 1 has highest priority in registration, and let fn_i' be an extension of fn_i in such a way that it contains n tf-pairs. Then the pair $occurs_i'$, fn_i' does the same for occurrence 1 in registration as the pair $occurs_i$, fn_i.

This principle works for every pair of an fn and an n-tuple of occurrence numbers, which leads to a number of ordering principles for the $2^n - 1$ arrays of occurrence numbers if all of them, and all fn_i, are extended to contain n occurrence numbers and n tf-pairs respectively.

- there have to be $\binom{n}{1}$ ordered n-tuples of occurrence numbers that each have a different occurrence number at the beginning of the ordered n-tuple; and
- there have to be $\binom{n}{2}$ ordered n-tuples of occurrence numbers that each have a different combination of two occurrence numbers at the beginning of the ordered n-tuple; and

.

.

- there have to be $\binom{n}{n} = 1$ ordered n-tuple that has n occurrence numbers at the begining of the ordered n-tuple (this could be any node, since all have n labels).

Using these principles the number of fn's of a tree structure can be reduced, because an n-tuple of n occurrence numbers contains one combination of one ocurrence number, one combination of two occurrence numbers, etc. The total number of fn's of a tree structure then amounts to comb(n), where n is the capacity of the tree structure, since comb(n) is the maximum number of combinations with an equal number of elements, for a given n.

The reduction is realised by ordering comb(n) n-tuples of occurrence numbers in such a way that all combinations have highest priority at least once. This can be done using the Ordering Method, a description of which is preceded by a definition

of an ordered m-tuple of ordered n-tuples of occurrence numbers.

Definition 26 (of $OCCURS_t$)

Let t = ‹root, final nodes› be a tree structure of capacity n; $OCCURS_t$ is an ordered m-tuple of ordered n-tuples (called $occurs_i$, i = 1, . . , comb(n)) of occurrence numbers of t, ordered according to the Ordering Method.

Ordering Method:

- t is a tree structure of capacity n, m = comb(n);
- Let $OCCURS_t'$ be a randomly ordered m-tuple of randomly ordered n-tuples of the occurrence numbers of t;
- Let i increase from 1 to n-1;
  - for every i:
    - adjust the order of the occurrence numbers of the n-tuples to have every combination of i occurrence numbers at the beginning of the n-tuples as often as possible (so all combinations are at the beginning of an n-tuple about an equal number of times), respecting the order adjustments already made.

An example of an application of the Ordering Method:

let n=5, the n-tuples are written in columns, unordered occurrence numbers are printed cursively.

i increases from 1 to 5, $\max(\binom{5}{3}) = 10$, 10 n-tuples of occurrence numbers:

1	*5*	*3*	*4*	*4*	*2*	*3*	*1*	*2*	*5*
4	*2*	*5*	*3*	*1*	*5*	*2*	*4*	*3*	*1*
5	*1*	*2*	*1*	*2*	*4*	*5*	*3*	*4*	*3*
3	*3*	*4*	*5*	*5*	*1*	*4*	*2*	*1*	*2*
2	*4*	*1*	*2*	*3*	*3*	*1*	*5*	*5*	*4*

(i=1) $\binom{5}{1} = 5$, this gives the folowing occurrence numbers;

a	b	c	d	e
1	**2**	**3**	**4**	**5**

ordering the fn's results in:

1	**2**	**3**	**4**	**5**	**1**	**2**	**3**	**4**	**5**
4	*5*	*5*	*3*	*1*	*5*	*3*	*4*	*3*	*1*
5	*1*	*2*	*1*	*2*	*4*	*5*	*1*	*2*	*3*
3	*3*	*4*	*5*	*4*	*2*	*4*	*2*	*1*	*2*
2	*4*	*1*	*2*	*3*	*3*	*1*	*5*	*5*	*4*

(i=2) $\binom{5}{2} = 10$ the combinations of occurrence numbers are:

1	**1**	**1**	**1**	**2**	**2**	**2**	**3**	**3**	**4**
2	**3**	**4**	**5**	**3**	**4**	**5**	**4**	**5**	**5**

the order adjustment results in:

1	2	3	4	5	1	2	3	4	5
2	3	4	5	1	3	4	5	1	2
5	*1*	*2*	*1*	*2*	*4*	*5*	*1*	*2*	*3*
3	*5*	*5*	*3*	*4*	*2*	*3*	*2*	*3*	*1*
4	*4*	*1*	*2*	*3*	*5*	*1*	*4*	*5*	*4*

(i=3) $\binom{5}{3} = 10$, the combinations are:

1	1	1	1	1	1	2	2	2	3
2	2	2	3	3	4	3	3	4	4
3	4	5	4	5	5	4	5	5	5

the order adjustments result in:

1	2	3	4	5	1	2	3	4	5
2	3	4	5	1	3	4	5	1	2
3	4	5	1	2	4	5	1	2	3
5	*5*	*2*	*3*	*4*	*2*	*3*	*2*	*3*	*1*
4	*1*	*1*	*2*	*3*	*5*	*1*	*4*	*5*	*4*

(i=4) $\binom{5}{4} = 5$ the combinations are:

1	1	1	1	2
2	2	2	3	3
3	3	4	4	4
4	5	5	5	5

adjustment results in:

1	2	3	4	5	1	2	3	4	5
2	3	4	5	1	3	4	5	1	2
3	4	5	1	2	4	5	1	2	3
4	5	1	2	3	5	1	2	3	4
5	*1*	*2*	*3*	*4*	*2*	*3*	*4*	*5*	*1*

which are ordered columns of five numbers:

1	2	3	4	5	1	2	3	4	5
2	3	4	5	1	3	4	5	1	2
3	4	5	1	2	4	5	1	2	3
4	5	1	2	3	5	1	2	3	4
5	1	2	3	4	2	3	4	5	1

Registration, see definition 28, boils down to an assignment of the value of an α_i variable of the root, denoted by an occurence number, to one of the atom-variables of a tf-pair of an fn. Since not all variables in the root are α_i variables, a function will be used that adresses only α_i variables.

Definition 27 (of an address function)

Let $t = \langle root, final\ nodes \rangle$ be a tree-structure of capacity n,

$root = \langle x_1, x_2, \ldots, x_j \rangle$, and

$occurs_i = \langle o_1, o_2, \ldots, o_n \rangle$ is an ordered n-tuple of the occurrence numbers of t;

$ADD : \{o \mid o \text{ in } occurs_i\} \rightarrow \{1, \ldots, j\}$ is a function that delivers the index m, $1 \leq m \leq j$, of an atom-variable x in the root; $ADD(o_k) = o_k.n + o_k - 1$, $k \in \{1, \ldots, n\}$;

The registration function is defined over the several structural layers of final nodes and OCCURS. Registration of an occurrence takes place if it is placed in disjunction with all occurrences, that have higher priority in that fn, and that can be registered themselves. An occurrence is registered in the first (leftmost or top-most) atom-variable of a tf-pair if the unary-operator-variable immediately left of it in the root has the value $\sim$, otherwise it is registered in the second atom-variable, if it is registered at all.

Definition 28 (of registration)

let $t = \langle root, final\ nodes \rangle$ be a tree structure of capacity n;

$root = \langle x_1, x_2, \ldots, x_j \rangle$; final nodes $= \langle fn_1, fn_2, \ldots, fn_m \rangle$;

$fn_i = \langle\langle y_{11}, y_{12} \rangle, \langle y_{21}, y_{22} \rangle, \ldots, \langle y_{n1}, y_{n2} \rangle\rangle$, $i \in \{1, \ldots, m\}$;

$OCCURS_t = \langle occurs_1, occurs_2, \ldots, occurs_m \rangle$;

$occurs_i = \langle o_1, o_2, \ldots, o_n \rangle$, $i \in \{1, \ldots, m\}$;

registration is a function $REG : \{root\} \times \{final\ nodes\} \times \{OCCURS_t\} \rightarrow \{final\ nodes\}$;

(i) $REG(root, final\ nodes, OCCURS_t) =$

$REG(root, \langle fn_1, fn_2, \ldots, fn_m \rangle, \langle occurs_1, occurs_2, \ldots, occurs_m \rangle) =$

$\langle REG(root, fn_1, occurs_1), REG(root, fn_2, occurs2), \ldots, REG(root, fn_m, occurs_m) \rangle$;

(ii) $REG(root, fn_p, occurs_p) =$

$REG(root, \langle\langle y_{11}, y_{12} \rangle, \langle y_{21}, y_{22} \rangle, \ldots, \langle y_{n1}, y_{n2} \rangle\rangle, \langle o_1, o_2, \ldots, o_n \rangle) =$

$\langle REG(x_{ADD(o_1)}, \langle y_{11}, y_{12} \rangle, o_1), REG(x_{ADD(o_2)}, \langle y_{21}, y_{22} \rangle, o_2), \ldots,$

$REG(x_{ADD(o_n)}, \langle y_{n1}, y_{n2}, o_n \rangle \rangle$;

(iii) $REG(x_{ADD(o_q)}, \langle y_{q1}, y_{q2} \rangle, o_q) =$

(a) $\langle x_k, * \rangle$, where $k = ADD(o_q)$, if $x_k \neq *$, and for all x_r such that:

(i) $REG(x_{ADD(o_s)}, \langle y_{s1}, y_{s2} \rangle, o_s)$ is $\langle *, x_r \rangle$ or $\langle x_r, * \rangle$; and

(ii) $s < q$,

that occur in fn_p: $D_{R(root)}(R(x_r), R(x_k))$ and $N_{R(root)}(R(x_k))$ hold.

(b) $\langle *, x_k \rangle$, where $k = ADD(o_q)$, if $x_k \neq *$ and for all x_r such that:

(i) $REG(x_{ADD(o_s)}, \langle y_{s1}, y_{s2} \rangle, o_s)$ is $\langle *, x_r \rangle$ or $\langle x_r, * \rangle$; and

(ii) $s < q$,

that occur in fn_p: $D_{R(root)}(R(x_r), R(x_k))$ holds and $N_{R(root)}(R(x_k))$ doesn't hold;

(c) $\langle *, * \rangle$ otherwise.

Theorem 7:

Let $\langle \varphi, PCES \rangle$ be a PT, $t = \langle root, final\ nodes \rangle$ is a tree structure; if $R(root) = \varphi$, then for each $PCE \in PCES$, $\langle TAT, FAT \rangle$, there is an fn_i in final nodes and an $occurs_i$ in $OCCURS_t$ such that $R(REG(root, fn_i, occurs_i)) = \langle TAT, FAT \rangle$.

Proof: see appendix I.

Closures and tree closures are not parts of a tree structure; they do not have representations that are parts of a parallel tableau. Closures and tree closures are functions meant to make reading a parallel tableau easier.

Definition 29 (of a closure)

Let $t = \langle root, final\ nodes \rangle$ be a tree structure of capacity n; a closure is a function $cl: \{fn \mid fn \text{ in final nodes}\} \rightarrow \{0, 1\}$;

$cl(fn) = 1$ if and only if there are x_j, x_k in fn such that $C_{R(root)}(R(x_j), R(x_k))$ holds;

0 otherwise.

Definition 30 (of tree closure)

Let $t = \langle root, final\ nodes \rangle$ be a tree structure of capacity n,

final nodes $= \langle fn_1, \ldots, fn_{comb(n)} \rangle$;

tree closure is a function

$tc: \{final\ nodes\} \rightarrow \{$"the formula is invalid", "the formula is valid"$\}$;

$tc(final\ nodes) =$ "the formula is valid" if and only if $comb(n) = \sum_{i=1}^{comb(n)} cl(fn_i)$;

"the formula is invalid" otherwise.

Chapter 4

Implementation of the Parallel Tableau Method in a Network of Perceptrons

The structure of a perceptron network in which the parallel tableau method is implemented is determined by the parallel tableau method. The parallel tableau method may be thought of as consisting of two parts, a passive part, the parallel tableau, which is a representation of a tree structure, and an active part, the parallel method, which consists of the tableau predicates at the level of parallel tableaux, and the registration, the closure, and the tree closure functions at the level of tree structures.

A tree structure is represented by perceptron predicates (idealised neurons) using a representation method The systematical representing of parts of a tree structure by perceptron predicates is the subject of paragraph two.

With the exception of the "negated" predicate, the tableau predicates are represented by dedicated perceptron predicates. The computation of the tableau predicates and the functions is embedded in the pattern of connections of the perceptron network. A very clear case is the registration function, where the registration heuristic, described extensively in chapter 3, is completely embedded in the connection pattern. The representation of the parallel method by perceptron predicates and the connection pattern of a perceptron network is discussed in paragraph three.

The structure of the perceptron network, and the parallel tableau method specify input - output relations for the separate perceptrons, in virtue of which the connection weights of the perceptrons that make up the network can be specified. This is the subject of paragraph four. Paragraph one is an introduction to perceptrons.

4.1 Perceptrons

An extensive description of perceptrons can be found in [Minsky & Papert 1969].

Definition 31 (of perceptron predicates)

Let $\varphi_1, \ldots \varphi_n, \psi$ be functions $\varphi_i : \mathbb{Z} \rightarrow \{0, 1\}$; and

$\psi : \mathbb{Z} \rightarrow \{0, 1\}$;

the functions φ_i, and ψ are called perceptron predicates.

Definition 32 (of $\psi(X)$)

Let $X \subset \mathbb{Z}$, $\Phi = \{\varphi_1, \varphi_2, \ldots, \varphi_n\}$; Θ is a number called the threshold; $\{\alpha_1, \alpha_2, \ldots, \alpha_n\}$ is a set of numbers called the weigths; $\psi(X) = 1$ if and only if $\sum_i \alpha_i \varphi_i(X) > \Theta$.

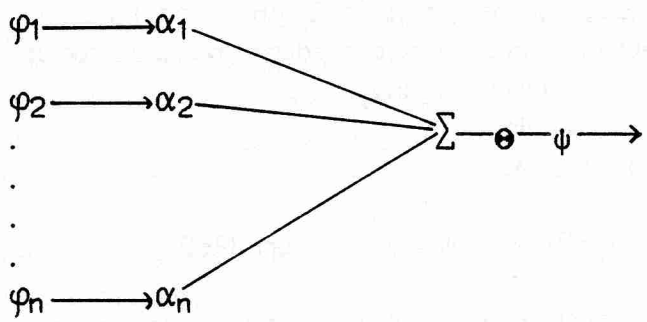
Definition 33 (of a perceptron)

Let ψ, φ_i, $i = 1, \ldots, n$, be a number of perceptron predicates, and let $X \subset \mathbf{Z}$; a perceptron is a computing device that computes $\psi(X)$ in two stages:

(i) compution of the $\varphi_i(X)$, in a mutually independent way; and

(ii) determination of $\psi(X)$ as in definition 32.

fig 1: a perceptron.



Let $\mathbf{A}$ denote a row vector of the elements of $\{\alpha_1, \ldots, \alpha_n\}$ ordered in a arbitrary but fixed way. Let $\mathbf{\Phi}(X)$ denote a row vector of the elements of $\{\varphi_1(X), \ldots, \varphi_n(X)\}$ ordered in the same way as $\mathbf{A}$. $\psi(X) = 1$ if and only if $\mathbf{A} . \mathbf{\Phi}(X) > \Theta$, where $\mathbf{A} . \mathbf{\Phi}(X)$ is the vector inproduct of $\mathbf{A}$ and $\mathbf{\Phi}(X)$.

The weights of a perceptron are determined with the aid of the perceptron convergence procedure.

Definition 34 (of the perceptron convergence procedure)

Let $F^-, F^+ \subseteq \mathbf{Z}$, $X \subseteq F^+$ or $X \subseteq F^-$, and $F^+ \cap F^- = \emptyset$; if $X \subseteq F^+$ then $\psi(X)$ should yield the value 1, if $X \subseteq F^-$ then $\psi(X)$ should yield the value 0; the perceptron convergence procedure is given the form of the following (pseudo) computer program [Minsky and Papert 1969]:

START: Choose any value for $\mathbf{A}$.

TEST: Choose any X from $F^+ \cup F^-$.
if $X \subseteq F^+$ and $\mathbf{A} . \mathbf{\Phi}(X) > 0$ go to TEST.
if $X \subseteq F^+$ and $\mathbf{A} . \mathbf{\Phi}(X) \leq 0$ go to ADD.
if $X \subseteq F^-$ and $\mathbf{A} . \mathbf{\Phi}(X) < 0$ go to TEST.
if $X \subseteq F^-$ and $\mathbf{A} . \mathbf{\Phi}(X) \geq 0$ go to SUBTRACT.

ADD: Replace $\mathbf{A}$ by $\mathbf{A} + \mathbf{\Phi}(X)$.
Go to TEST.

SUBTRACT: Replace $\mathbf{A}$ by $\mathbf{A} - \mathbf{\Phi}(X)$.
Go to TEST.

The procedure stops if the α_i do not change any more. The perceptron convergence theorem assures us that in that case we have found weights that make the perceptron operate properly.

Theorem 8 (perceptron convergence theorem):
If there exists a vector $\mathbf{A}^*$ such that if
$X \subseteq F^+$ then $\mathbf{A}^* . \mathbf{\Phi}(X) > 0$, and if
$X \subseteq F^-$ then $\mathbf{A}^* . \mathbf{\Phi}(X) \leq 0$
then, whatever the choices made for $\mathbf{A}$ in START, and for X in test, $\mathbf{A}$ will be changed, by the perceptron convergence procedure, into a vector $\mathbf{A}^o$ such that:
if $X \subseteq F^+$ then $\psi(X) = 1$, and
if $X \subseteq F^-$ then $\psi(X) = 0$,
within a finite number of changes.

Proof: see [Rosenblatt 1962] or [Minsky & Papert 1969].

Notice that only the $\varphi_i(X)$ are relevant in determining the α_i, not the functions φ_i themselves. The essence of determining the α_i is that there are vectors $\mathbf{\Phi}(X)$ for which $\psi(X)$ should be 1, and vectors $\mathbf{\Phi}(X)$ for which $\psi(X)$ should be 0. The binary versions of the former will be called $\mathbf{F}^+$ vectors, the binary versions of the latter will be called $\mathbf{F}^-$ vectors. Clearly if $X \subseteq F^+$ then $\mathbf{\Phi}(X)$ is an $\mathbf{F}^+$ vector, and similarly if $X \subseteq F^-$ then $\mathbf{\Phi}(X)$ is an $\mathbf{F}^-$ vector. We can therefore replace "$X \subseteq F^+$" and "$X \subseteq F^-$" by "$X \Rightarrow \mathbf{F}^+$" and "$X \Rightarrow \mathbf{F}^-$" respectively in the perceptron convergence procedure, and the perceptron convergence theorem without destroying the adequacy of either of them. The "$\Rightarrow$" symbol should be read as "implies".

The parallel tableau method will be implemented in a network of connected perceptrons; the ψ predicates will function as φ_i predicates in other perceptrons. This network should not be interpreted as one multilayer perceptron, but as several connected perceptrons. The difference is that the weights of a multilayer perceptron cannot, but the weights for each perceptron in a network of connected perceptrons, can be determined using the perceptron convergence procedure, if the $\mathbf{F}^+$'s and the $\mathbf{F}^-$'s are known.

When evolving the network, all predicates, except the predicates representing the input, are updated at once (synchronously), this way the neural activity represented by the input spreads through the entire network. Notice that there is no feed back in perceptrons or perceptron networks, that is why perceptrons are called feed forward networks.

4.2 The Representation of a Tree Structure

The values of the variables that make up a tree structure are representations of arrays of, and single, binary digits. A binary digit is the value of a perceptron predicate. Representation of tree structure variable values by binary codes is the

subject of paragraph 4.2.1, the representation of parts of a tree structure by perceptron predicates is discussed in paragraph 4.2.2.

4.2.1 Values of Tree Structure Variables as Binary Codes

Since tree structures are represented by perceptron predicates, all values of the tree variables have to be represented by binary codes, arrays of binary digits. The binary codes are defined, in general, in definition 35. Before proceeding with the definition it should be noted that in every finite implementation of the parallel tableau method only a limited number of atoms will be available.

AV is the alfabetically ordered set of atoms available in some implementation of the parallel tableau method, and #(AV) is the number of elements of AV.

Definition 35 (of binary representation of the values of tree structure variables)

Let t be a tree structure, then:

(i) an atom has a binary representation equal to 2^i ; $i \in \{0, 1, \ldots, \#(AV) - 1\}$ in an array of #(AV) binary digits, where the index denotes the $(i+1)$-th element of AV;

(ii) a "$\sim$" is represented as 1 by one binary digit;

(iv) a "v" is represented as 1 by one binary digit, and an "&" is represented as 0 by the same digit;

(iv) a "(" and a ")" are represented as 1 by one binary digit.

Example 3,

binary representation of an instantiation of a tree structure of capacity 3, where AV = {A,B,C}. The tree structure represents an analysis of A v (B & $\sim$ B);

values of the root:

$(_1$ $\sim_1$ α_1 $\square_1$ $(_2$ $\sim_2$ α_2 $)_1$ $\square_2$ $\sim_3$ α_3 $)_3$

binary representation:

0 0 001 1 1 0 010 0 0 1 010 1

binary representation of values in the fn's:

fn_1		fn_2		fn_3	
000	001	000	010	010	000
000	010	000	000	000	001
000	000	000	001	000	000

4.2.2 Binary Values and Perceptron Predicates

The binary values are the two values perceptron predicates can have. The representation of perceptrons in hardware or when simulated by computer is a matter that will be ignored. In this paper perceptrons are the rock bottom concerning representational layers.

To be able to conveniently and uniquely identify perceptron predicates, each predicate will be given an index that shows what the predicate represents in the network. This way kinds of perceptron predicates are defined.

Definition 36 (of kinds of perceptron predicates)

Let t be a tree structure of capacity n, $p = \#(AV)$:

(i) $\varphi_{r,i,j}$ an array of p φ_r predicates represents atom-variable α_i (in the root of a tree structure) $i \in \{1, \ldots, n\}$, $j = 1, \ldots, p$;

(ii) $\varphi_{fn,i,m}$ an array of p φ_{fn} predicates represents a tf-pair in a fn of a tree structure in which occurrence i can be registered; $i \in \{1, \ldots, n\}$, $m = 1, \ldots, 2p$, if $1 \leq m \leq p$ the predicate represents the leftmost atom-variable in a tf-pair;

(iii) $\varphi_{\sim,i}$ $i \in \{1, \ldots, n\}$, a $\varphi_{\sim,i}$ predicate represents a unary-operator-variable;

(iv) $\varphi_{\square,i}$ $i \in \{1, \ldots, n-1\}$, a $\varphi_{\square,i}$ predicate represents a binary-operator-variable;

(v) $\varphi_{(,i}$ index i is used to identify pairs of rpar- and lpar-variable representing predicates, a $\varphi_{(,i}$ predicate represents an lpar-variable;

(vi) $\varphi_{),i}$ see also (v), $\varphi_{),i}$ represents an rpar-variable;

(vii) φ_{Th} this predicate is used to add an extra threshold to some perceptrons, it is a hidden unit and for all $X \subset \mathbb{Z}$: $\varphi_{Th}(X) = 1$;

(viii) $\varphi_{D(i,j),k}$ this hidden unit represents an "in-disjunction" tableau predicate, i and j denote atom-variables α_i and α_j, k denotes a binary-operator-variable, if $i < j$ then $i \in \{1, \ldots, n-1\}$, $j \in \{2, \ldots, n\}$, and $k = i, \ldots, j-1$, if $\varphi_{D(i,j),k}(X) = 1$, then occurrences i and j are placed in disjunction, where k is the responsible operator;

(ix) $\varphi_{f,i,j}$ a φ_f predicate is a hidden unit needed to compute values of the registration function, i and j denote atom-variables α_i and α_j, $\varphi_{f,i,j}(X) = 1$ if in some fn occurrence j is registered and occurrence i is placed in-disjunction with occurrence j, $i,j \in \{1, \ldots, n\}$;

(x) $\varphi_{c,i}$ φ_c predicates are used to compute whether a certain atom in a fn causes that fn to be closed, $i \in \{1, \ldots, p\}$, there are p φ_c predicates per fn, φ_c predicates are hidden units;

(xi) $\varphi_{cl,i}$ φ_{cl} predicates represent closures, $\varphi_{cl,i}(X) = 1$ if at fn_i at least one $\varphi_{c,j}(X) = 1$, $i \in \{1, \ldots, \max(n,k)\}$;

(xii) φ_{tc} φ_{tc} represents a tree closure, it indicates whether all φ_{cl} predicates have the value 1.

4.3 Representing the Parallel Method

In this paragraph the structure of a perceptron network, being mainly the pattern of connections that constitute the computations of, among other things, the various tableau predicates, will be described.

Connections are metaphores for computational dependency. If the value of some predicate $\psi(X)$ depends on the value of another predicate $\varphi_i(X)$, because it delivers a part of the weighted sum that determines the value of $\psi(X)$, then there is a

connection from φ_i to ψ. The "from ... to ..." clause indicates the dependency relation. Connections can also be thought of as wires in a hardware implementation of a perceptron.

To specify connections in a network of perceptrons, in which the parallel tableau method is implemented, is not the same as to specify the perceptrons the network consists of, because several computations may be done in one perceptron.

4.3.1 Computation of the "In-Disjunction" Predicate

to determine whether two atom occurences are placed in disjunction, one needs to find the principal operator that has one occurrence as a subformula in the one argument, and the other occurrence as a subformula in the other argument. If this operator is an "or" operator, the occurrences are placed in disjunction. The determination whether two atom occurrences are placed in disjunction depends solely on structural properties of the formula under investigation, i.e. information concerning the binary operators and parentheses.

To compute whether two atom occurrences that have only one operator in between are placed in disjunction, only a connection from the $\varphi_{\Box,i}$ predicate to the $\varphi_{D(i,j),k}$ predicate is needed. For instance in:

fig. 2, the $\sim_i$ variables are left out for simplicity.

fragment of a simplfied root:

$$\ldots\ldots\ldots\ldots\ \alpha_3\ )_1\ )_2 \quad \Box_3 \quad (_3\ (_5\ (_6\ \alpha_4 \quad \ldots\ldots$$

representing predicate:

$\varphi_{\Box,3}$

$\downarrow$

$\varphi_{D(3,4),3}$

"in-disjunction" predicate.

it is sufficient that $\Box_3$ has the value "v" to have D(3,4) to hold. It is irrelevant whether one of the parentheses is present or not. Compare:

$\chi\&(\varphi\vee\psi)$; $D(\varphi,\psi)$ holds, with
$(\chi\&\varphi)\vee\psi$; both $D(\varphi,\psi)$ and $D(\varphi,\chi)$ hold.

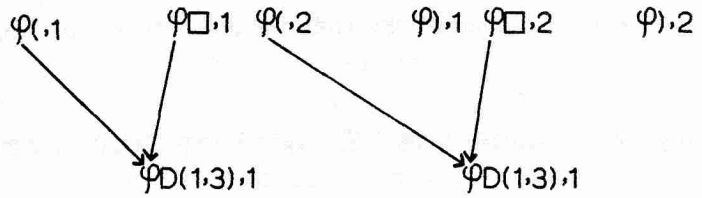
To compute which one of two operators is the principal operator of the entire formula requires connections from a predicate that represents a binary operator and the predicates that represent left parentheses that enclose the operator and precisely one of the occurrences, of which it is to be determined whether the "in-disjunction" predicate holds. The illustration below elucidates the case for n=3.

fig. 3, the $\sim_i$ variables are left out for simplicity.

a simplified root:

$(_1 \quad \alpha_1 \quad \Box_1 \quad (_2 \quad \alpha_2 \quad)_1 \quad \Box_2 \quad \alpha_3 \quad)_2$

predicates:



"in-disjunction" predicates.

- If $\varphi_{D(1,3),1}(X) = 1$ it follows that D(1,2) holds, this conclusion is left out to prevent redundancy in information distribution, which would make things more complicated.
- D(1,2) and D(2,3) are computed the way shown in fig. 2.
- Note that it is impossible that: $\varphi_{D(1,3),1}(X) = \varphi_{D(1,3),2}(X) = 1$, since both operators would have to be principal operators.

The scope of a pair of parentheses is equal to the number of binary operators enclosed by the pair of parentheses.

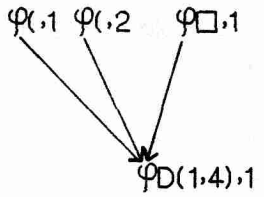
Computing whether the left operator (see below) is both the principal operator of the entire formula and an "or" operator, requires connections from one of each pair $\varphi_{(,i}$ and $\varphi_{),i}$ predicates that represent parentheses in which scope the left operator-variable occurs, and a binary-operator representing predicate. The computation for the right operator variable is the symmetrical variant.

fig. 4, the $\sim_i$ variables are left out for simplicity.

simplified root:

$(_1 \; (_2 \; \alpha_1 \quad \Box_1 \quad (_3 \; (_4 \; \alpha_2 \;)_2 \quad \Box_2 \quad (_5 \; \alpha_3 \;)_4 \;)_1 \quad \Box_3 \quad \alpha_4 \;)_5 \;)_3$

some predicates:



"in-disjunction" predicate.

- D(1,3) and D(2,4) are computed the way shown in fig. 3. In these cases parentheses variables with indices 1, and 3 do not play a part in the computation.
- D(1,2), D(2,3), and D(3,4) are computed the way shown in fig. 2.

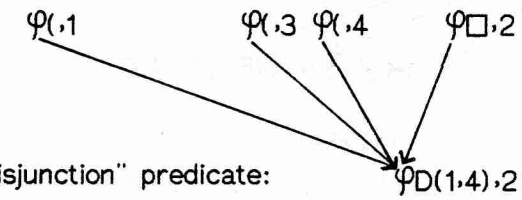
Computing whether the middle operator is the principal operator of the entire formula is similar to the previous case.

fig. 5, the $\sim_i$ variables are left out for simplicity.

simpified root:

$$(_1\ (_2\ \alpha_1 \quad \Box_1 \quad (_3\ (_4\ \alpha_2\ )_2 \quad \Box_2 \quad (_5\ \alpha_3\ )_4\ )_1 \quad \Box_3 \quad \alpha_4\ )_5\ )_3$$

some predicates:



- If $\varphi_{D(1,4),2}(X) = 1$, the formula under investigation has to have the form (1 □ 2) v (3 □ 4).

The number of $\varphi_{D(i,j),k}$ predicates in an implementation of the parallel tableau method depends on the capacity n of the implemented tableau. If $m = n+1$, and $n > 3$ then the number of $\varphi_{D(i,j),k}$ predicates is comb(m).

4.3.2 Computation of the "Negated" Predicate

The tableau predicate "negated" holds of an atom occurrence φ if and only if φ is a subformula of $\sim\varphi$.

The "negated" tableau predicate is not represented by a dedicated perceptron predicate. The $\varphi_{fn,i,j}$ predicates do double duty; if $\varphi_{fn,i,j}(X) = 1$ the tableau predicate is represented as holding when $1 \leq j \leq p$, and as not holding when $p < j \leq 2p$.

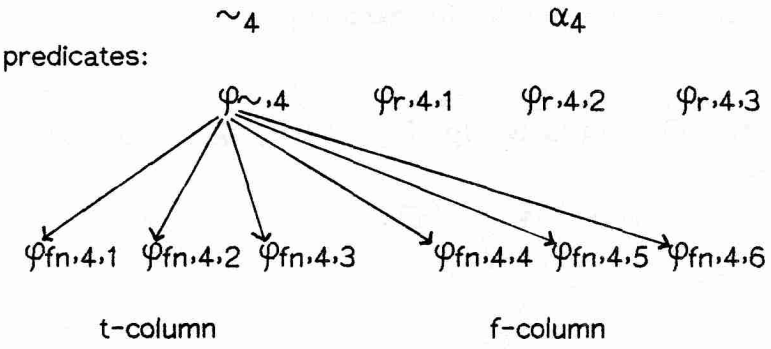
Let $\langle \langle x_{11}, x_{12} \rangle,$
$\langle x_{21}, x_{22} \rangle,$
.
.
$\langle x_{n1}, x_{n2} \rangle \rangle$

be a final node, it will be convenient to call $x_{11}, x_{21}, \ldots, x_{n1}$ the t-column and $x_{21}, x_{22}, \ldots, x_{n2}$ the f-column of a fn; an atom occurrence is registered as a value of a x_{i2} atom-variable if it is not negated in the formula under analysis, i.e. if the occurrence is a part of a counter-example it should be interpreted as having the value **f**.

The computation of the value of the "negated" predicate requires connections from a $\sim_i$ variable representing predicate to al predicates $\varphi_{fn,i,j}$; $j = 1, \ldots, 2p$ that represent a tf-pair in a fn. Figure 6 illustrates the matter.

fig. 6, #(AV) = 3:

fragment of a root:



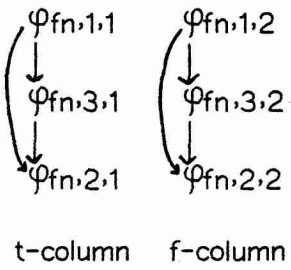
fragment of a fn.

4.3.3 Computation of the "Closed" Predicate

Before adressing the implementation of the "closed" predicate, attention should be paid to a closely related subject. In the implementation of the parallel tableau method sofar, the only limit to the number of ocurrences of the same atom that can be registered in a fn, is the number of tf-pairs of the fn. There is, however, no need to register more than one occurrence of an atom in one column of a fn. For esthetical reasons, and to make the implementation simpler, registration of more than one occurrence of the same atom in one column will be disabled, which requires connections as illustrated below.

fig. 7, #(AV) = 1, the capacity of the tree structure is 3.

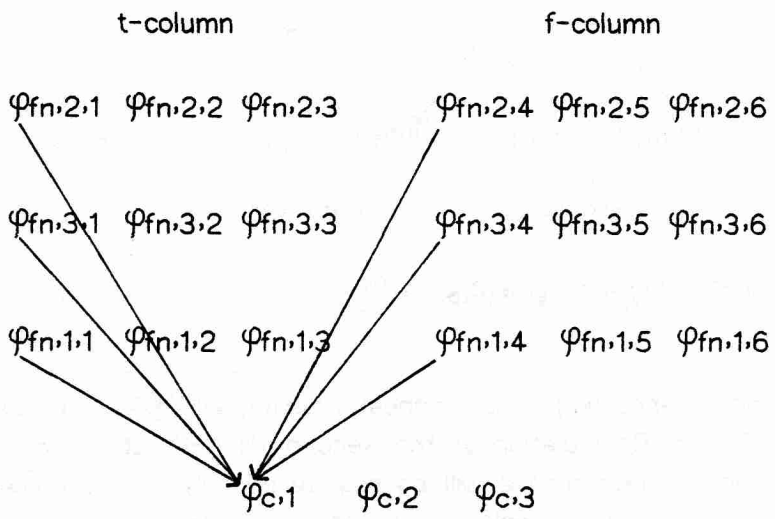
predicates representing a fn:



A PCE in a parallel tableau is closed if at least one atom occurs in both sets of the PCE. A tree structure comes with a closure for each fn. In a perceptron network, the value of a closure is determined in two steps. First it is determined for each atom whether it occurs in the t-column as well as in the f-column of the same fn, then it is determined whether there is at least one such an atom in an fn, which is determination of the value of a closure.

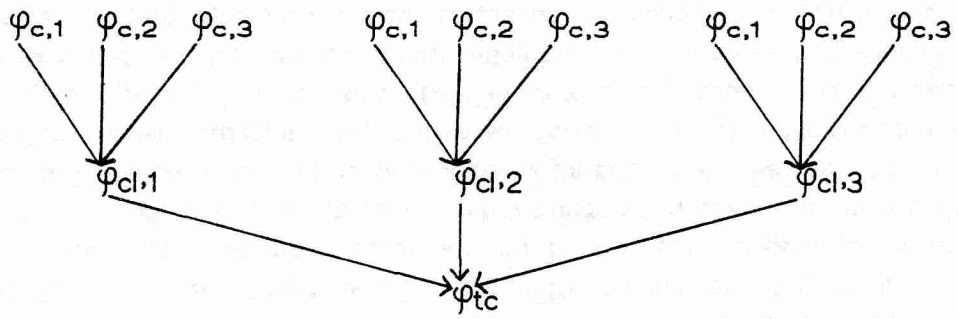
For the first step $\varphi_{c,j}$ predicates are used. There are connections from each $\varphi_{fn,i,j}$, and $\varphi_{fn,i,(j+p/2)}$ predicate $j \in \{1, \ldots, \#(AV)\}$ of a fn, which has the value 1 when a certain atom is represented by the array either as negated or not-negated, to a $\varphi_{c,j}$ predicate, see figure 8.

fig. 8, fragment of a fn, $\#(AV) = 3$, the capacity of the tree structure is 3.



All φ_c predicates of a fn are connected to a φ_{cl} predicate which represents a closure, and all φ_{cl} predicates are connected to one φ_{tc} predicate which represents a tree closure. Figure 9 elucidates the matter.

fig. 9.



4.3.4 Structural Requirements for Registration of Atom Occurrences in a FN

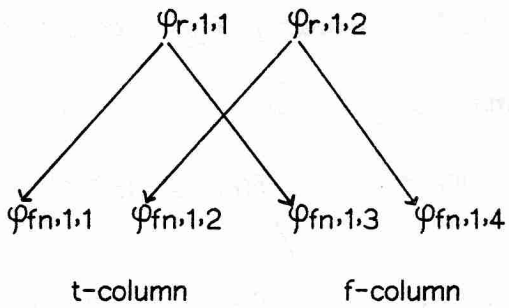
The registration of an atom occurrence in a fn requires connections from the atom-variable representing predicates in the root of the tree structure to the corresponding perceptron predicates in the fn's, as indicated by figure 10.

fig. 10, fragment of the root, #(AV) = 2.

atom-variable in the root:

α_1

perceptron predicates:



predicates representing a fragment of a fn.

The atom occurrence which has highest priority, will always be registered in the first tf-pair of a fn. Registration of the second till n-th atom occurrence is more complicated. Such an occurrence will be registered only if it is placed in disjunction with ocurrences with higher priority, if these occurrences have been registered themselves, so computing whether the second till n-th occurrence can be registered is recursive.

Luckily there is a shortcut. Instead of computing whether all occurences that have a higher priority than the occurrence that needs to be registered, satisfy all their conditions, it is sufficient to find out whether these occurrences are present in the fn, that is a certain sign that they do satisfy all their conditions.

Registration of the second till n-th occurrences requires hidden units. The function of such a hidden unit is to couple a connection from a predicate that indicates that some occurrence is present in a fn, and connections from perceptron predicates that indicate that the occurrence that has to be registered is placed in disjunction with the present occurrence. If this coupling were not done, information concerning the presence of occurrences in a fn and information concerning occurrences being placed in disjunction could be mixed up, causing faulty registration of ocurrences.

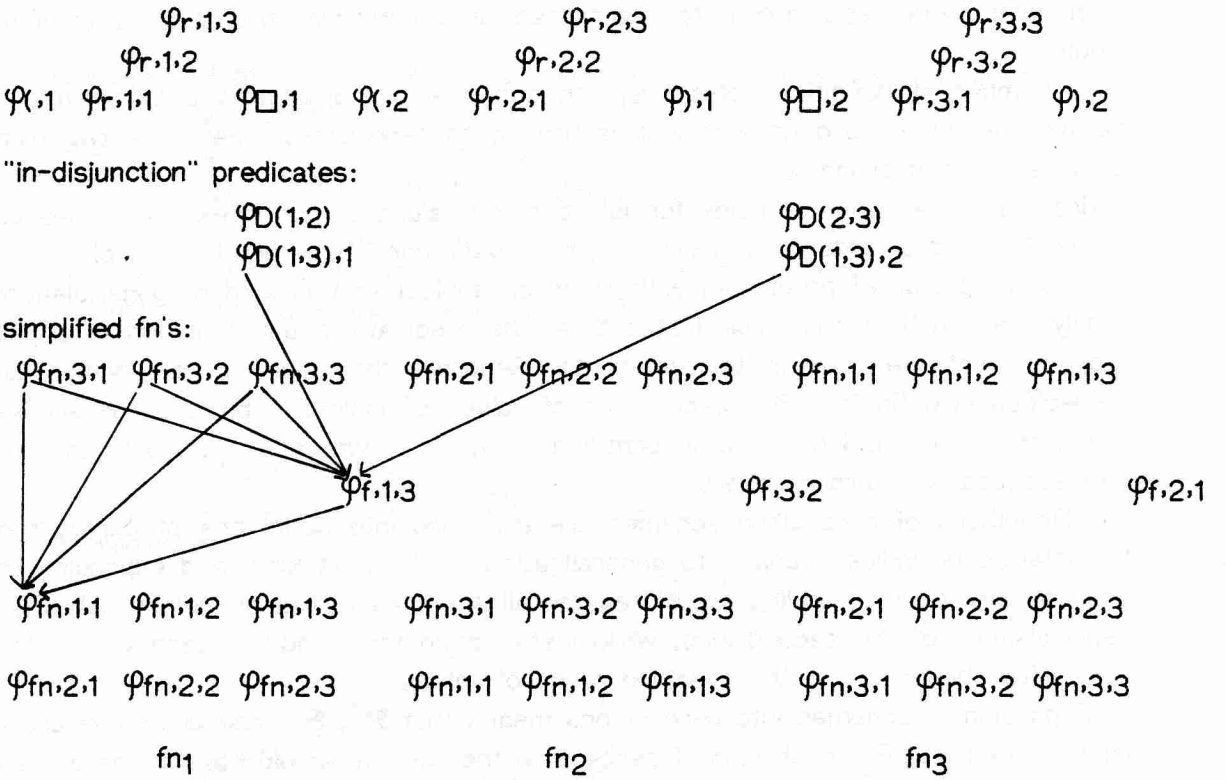
The number of hidden units needed for the m-th occurrence that needs to be registered is m-1. The total number of needed hidden units in an implementation of the parallel tableau method is:

$$\text{comb}(n).\sum_{i=1}^{n-1} i$$

Registration itself requires a connection from each hidden unit, that signals that the occurrence to be registered is effectively placed in disjunction with some occurrence with higher priority, and a connection from each predicate in each higher position that signals that some occurrence is present (or not). The matter is illustrated in figure 11.

fig. 11, registration in the second position of fn's, the capacity of the tree structure, and the number of available atoms is three; $\varphi_{\sim,i}$ predicates and t-columns are left out for simplicity.

3D representation of predicates representing a simplified root:



4.4 Perceptrons and their Coefficients

In this paragraph a description of the perceptron network is given by describing which predicates constitute the individual perceptrons, and how perceptrons are connected to make up the perceptron network. The coefficients of each perceptron are computed with the aid of the perceptron convergence procedure. but since the determination proces is not very interesting only two cases are written out in full, as examples.

More precisely: it will be told for each perceptron: 1) which predicate functions as the ψ predicate; 2) which predicates are connected to this ψ predicate: the vector $\mathbf{\Phi}(X)$ of $\varphi_i(X)$; 3) what the $\mathbf{F}^+$ vectors are; all other binary row vectors with the same number of elements as a $\mathbf{F}^+$ vector of a perceptron are $\mathbf{F}^-$ vectors for that perceptron; and 4) what the coefficients are.

In the perceptron network a so called threshold-value predicate, named φ_{Th}, will be frequently used. The need for this predicate emerges from the fact that for some perceptrons a vector $\mathbf{A}$ of coefficients cannot be found. For instance, let $\mathbf{F}^+$is [1 1], then there is no vector of weights $\mathbf{A}$ that distinguishes [1 1] from both [0 1] and [1 0]. Since weights have to be positive in this case, the weighted sums of the values

of the input vectors are all greater than nil. The problem is solved by adding φ_{Th} to such perceptrons. The perceptron convergence procedure will give it a negative coefficient, -1 in this case. It can be said that φ_{Th} serves as an additional, variable threshold. The value of the coefficient α_{Th} will be computed by first augmenting all $\mathbf{F}^+$ and $\mathbf{F}^-$ vectors with the value of the φ_{Th} predicate, as the last element, and then using the perceptron convergence procedure to determine the coefficients of the whole.

A number of definitions of perceptron schemes are given in the following subparagraphs. In these definitions indices have been generalised. There are two types of generalisation of indices:

- first type: the definition holds for all specified values of the index, the values are specified, for an index i, as $i \in \{1, \ldots, b\}$ in definition 36, where $\{1, \ldots, b\} \subset \mathbb{N}$;
- second type: of all predicates with an index subject to this kind of generalisation, only one can have the value 1 at a time, the predicate in the definition of a perceptron scheme denotes this predicate. Generally the values of an index i are specified in definition 36. Specification of values of indices subject to generalisation of the second kind takes a form like: i = a, . . ,b, where a, . . , b is an array of successive natural numbers.

Definitions of perceptron schemes are expanded into definitions of perceptrons by instantiating indices subject to generalisation of the first kind, and expanding the scheme definitions by adding predicates for all specified values of indices subject to generalisation of the second kind, while instantiating these indices each by a different value chosen out of the specified range of values.

Expansion of schemes into perceptrons means that $\mathbf{F}^+$, $\mathbf{F}^-$, and $\mathbf{A}$ vectors are to be expanded too. For each kind of perceptron the value it should have in the $\mathbf{F}^+$ and $\mathbf{A}$ vectors is specified. Expansion of the $\mathbf{F}^+$, $\mathbf{F}^-$, and $\mathbf{A}$ vectors should be done using the appropriate values.

4.4.1 Registration of Occurrences in the First Position of a FN

Definition 37 (of a perceptron scheme for registration in the first tf-pair of a fn)

Let t be a tree structure, $fn_q = \langle\langle x_{11}, x_{12}\rangle, \ldots \langle x_{n1}, x_{n2}\rangle\rangle$ is a fn of t, index i is subject to generalisation of the first kind, index j to generalisation of the second kind, $\psi = \varphi_{fn,i,j}$ is a predicate in an array representing $\langle x_{11}, x_{12}\rangle$ in fn_q;

$j = 1, \ldots, p$, $\Phi(X) = [\varphi_{\sim,i}(X)\ \varphi_{i,j}(X)\ \varphi_{Th}(X)]$, $\mathbf{F}^+ = [1\ 1\ 1]$, and, by perceptron convergence procedure, $\mathbf{A} = [1\ 1\ -1]$;

(ii) $j = p+1, \ldots, 2p$, $\Phi(X) = [\varphi_{\sim,i}(X)\ \varphi_{i,j}(X)]$, $\mathbf{F}^+ = [0\ 1]$, and, by perceptron convergence procedure, $\mathbf{A} = [-1\ 1]$.

Application of the perceptron convergence procedure to the first case:

Let $\mathbf{A} = [0\ 0\ 0]$, $\mathbf{F}^+ = [1\ 1\ 1]$; $\psi(X) = 0 \Rightarrow$(addition)

$\mathbf{A} = [1\ 1\ 1]$, $\mathbf{F}^- = [0\ 0\ 1]$; $\psi(X) = 1 \Rightarrow$(subtraction)

$\mathbf{A} = [1\ 1\ 0]$, $\mathbf{F}^- = [1\ 0\ 1]$; $\psi(X) = 1 \Rightarrow$(subtraction)

$\mathbf{A} = [0\ 1\ -1]$, $\mathbf{F}^+ = [1\ 1\ 1]$; $\psi(X) = 0 \Rightarrow$(addition)
$\mathbf{A} = [1\ 2\ 0]$, $\mathbf{F}^- = [0\ 1\ 1]$; $\psi(X) = 1 \Rightarrow$(subtraction)
$\mathbf{A} = [1\ 1\ -1]$.
The procedure has finished, the reader can verify that the coefficients will not change anymore.

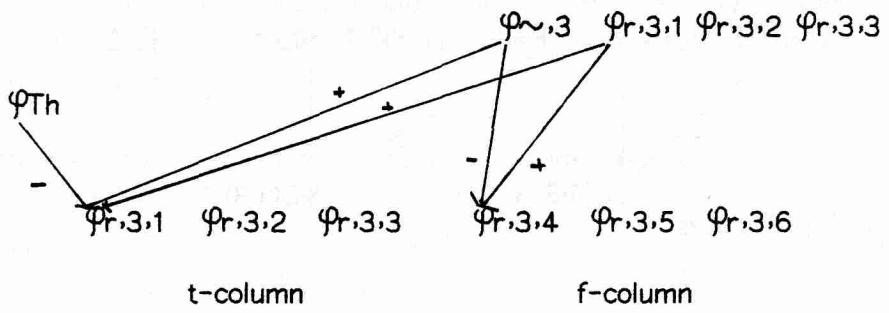
Application of the perceptron convergence procedure to the second case:
Let $\mathbf{A} = [0\ 0]$, $\mathbf{F}^+ = [0\ 1]$; $\psi(X) = 0 \Rightarrow$(addition)
$\mathbf{A} = [0\ 1]$, $\mathbf{F}^- = [1\ 1]$; $\psi(X) = 1 \Rightarrow$(subtraction)
$\mathbf{A} = [-1\ 0]$, $\mathbf{F}^+ = [0\ 1]$; $\psi(X) = 0 \Rightarrow$(addition)
$\mathbf{A} = [-1\ 1]$.
The procedure has finished since $\mathbf{A}$ discerns all $\mathbf{F}^+$ from all $\mathbf{F}^-$ in the correct way. Figure 12 illustrates the situation for $i = 3$.

fig. 12, #(AV) = 3.

fragment of a simplified root:

... $\sim_2$ α_2 $\square_2$ $\sim_3$ α_3 ....

some predicates:



fragment of a fn, first tf-pair.

4.4.2 Computation of the "In-Disjunction" Predicate

Definition 38 (of a perceptron scheme for the "in-disjunction" predicate)
Let $\psi = \varphi_{D(i,j),k}$, indices i and j are subject to generalisation of the first kind, m and k to generalisation of the second kind;

(i) if $j = i+1$: $\Phi(X) = [\varphi_{\square,i}(X)]$, $\mathbf{F}^+ = [1]$, and, by perceptron convergence procedure, $\mathbf{A} = [1]$; otherwise

(ii) $\Phi(X) = [\varphi_{(,m}(X)\ \varphi_{\square,k}(X)]$, $\mathbf{F}^+ = [0\ 1]$, and, by perceptron convergence procedure, $\mathbf{A} = [-1\ 1]$, index m may take on all values for which $\varphi_{(,m}$ denotes a lpar-variable in which scope the binary-operator-variable, and precisely one of the atom-variables denoted by i and j occur.

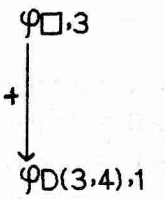
Figure 13 illustrates the first case, and figure 14 illustrates the second case.

fig. 13, the $\sim_i$ variables are left out for simplicity.

fragment of a simplified root:

..... α_3 $)_1$ $)_2$ $\square_3$ $(_3$ $(_4$ $(_5$ 4 .....

representing predicate:



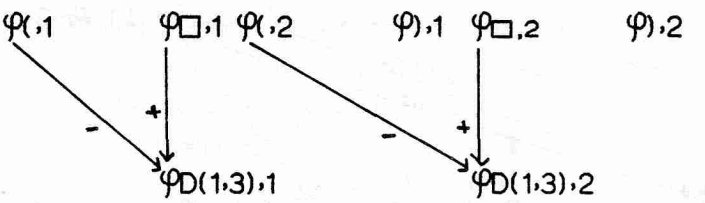
"in-disjunction" predicate.

fig. 14, the $\sim_i$ variables are left out for simplicity.

simplified root:

$(_1$ α_1 $\square_1$ $(_2$ α_2 $)_1$ $\square_2$ α_3 $)_3$

representing predicates:



"in-disjunction" predicates.

The scheme defined in definition 39 can be used for all cases in which "in-disjunction" predicates need to be computed. The coefficients are easy to determine, since every $\varphi_{(,i}(X) = 1$ predicate should prevent that $\varphi_{D(k,j),m}(X) = 1$ if both predicates are connected. So because a connection from $\varphi_{\square,m}$ to $\varphi_{D(k,j),m}$ has coefficient 1, al connections from $\varphi_{(,i}$ to $\varphi_{D(k,j),m}$ have coefficients -1.

4.4.3 Registration of Occurrences in the Second Position of a Final Node

Registration in the second till n-th tf-pair of a fn requires hidden units $\varphi_{f,i,j}$. Registration in the second tf-pair requires one hidden unit, in the third tf-pair two hidden units, etc.

Below, a definition of a perceptron scheme in which hidden unit $\varphi_{f,i,j} = \psi$ is given, followed by a perceptron scheme definition for registration in the second tf-pair of a fn. The subparagraph ends with a remark on schemes for perceptrons that embody registration in the third and further tf-pairs of a fn.

Definition 39 (of a perceptron scheme for hidden units in registration)

Let $\psi = \varphi_{f,i,j}$, indices i and j are subject to generalisation of the first kind, k and m to generalisation of the second kind;

$\mathbf{\Phi}(X) = [\varphi_{D(i,j),k}(X)\ \varphi_{fn,j,m}(X)\ \varphi_{Th}(X)]$, $\mathbf{F}^{+} = [1\ 1\ 1]$, and, by perceptron convergence procedure, $\mathbf{A} = [1\ 1\ -1]$.

Definition 40 (of a perceptron scheme for registration in the second tf-pair of a fn)

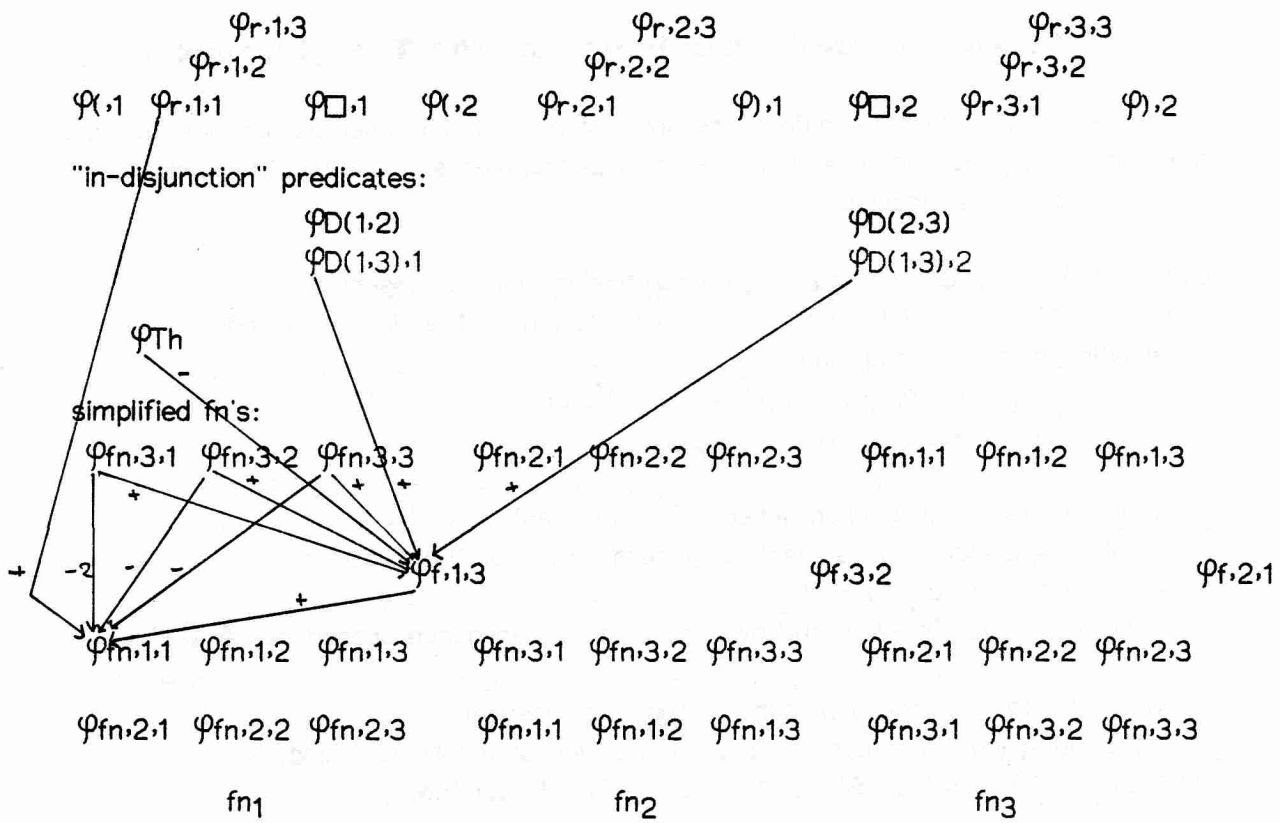
Let t be a tree structure, $fn_q = \langle\langle x_{11}, x_{12}\rangle, \ldots, \langle x_{n1}, x_{n2}\rangle\rangle$ is a final node of t, $\varphi_{fn,k,l}$ is a predicate of the array representing $\langle x_{11}, x_{12}\rangle$ in fn_q, $\psi = \varphi_{fn,i,m}$ is a predicate in an array representing $\langle x_{21}, x_{22}\rangle$ in fn_q, indices k, m, and i are subject to generalisation of the first kind, j and l ($l \neq m$) are subject to generalisation of the second kind;

(i) $j = 1, \ldots, p$, $\mathbf{\Phi}(X) = [\varphi_{\sim,i}(X)\ \varphi_{r,i,j}(X)\ \varphi_{f,i,k}(X)\ \varphi_{fn,k,l}(X)\ \varphi_{fn,k,m}(X)\ \varphi_{Th}(X)]$, $\mathbf{F}_1^{+} = [1\ 1\ 1\ 1\ 0\ 1]$, $\mathbf{F}_2^{+} = [1\ 1\ 0\ 0\ 0\ 1]$, and, by perceptron convergence procedure, $\mathbf{A} = [1\ 1\ 1\ -1\ -2\ -1]$;

(ii) $j = p+1, \ldots, 2p$, $\mathbf{\Phi}(X) = [\varphi_{\sim,i}(X)\ \varphi_{r,i,j}(X)\ \varphi_{f,i,k}(X)\ \varphi_{fn,k,l}(X)\ \varphi_{fn,k,m}(X)]$, $\mathbf{F}_1^{+} = [0\ 1\ 1\ 1\ 0]$, $\mathbf{F}_2^{+} = [0\ 1\ 0\ 0\ 0]$, and, by perceptron convergence procedure, $\mathbf{A} = [-1\ 1\ 1\ -1\ -2]$.

fig. 15, registration in the second position of fn's, the capacity of the tree structure, and the number of available atoms is three; $\varphi_{\sim,i}$ predicates and t-columns are left out for simplicity.

3D representation of predicates representing a simplified root:



The perceptron predicate $\varphi_{fn,k,m}$ is a predicate out of the array representing the first tf-pair. If this predicate has the value 1, the same occurrence is to be registered in one column of one fn again.

Registration in the second tf-pair of a fn can easily be generalised to cover registration in the third till n-th tf-pair. From the second tf-pair downward, each position requires one more hidden unit. In the case of registration in the second tf-pair $\varphi_{f,i,j}$ was used to couple connections that indicate presence in the first tf-pair and being placed in disjunction with the occurrence in the first tf-pair. Let's say that the hidden unit is concerned with the first tf-pair. Registration in the third tf-pair requires two hidden units, one that is concerned with the first tf-pair and one that is concerned with the second tf-pair, and so on. This way the connections and the coefficients are easily determined since all do the same.

In the $\Phi(X)$ vector each tf-pair in which an occurence is registered that has higher priority than the one to be registered, returns as a triplet of predicates. The first says something about being placed in disjunction, the second about presence in the earlier tf-pair, and the third whether the same atom is about to be registered again, see for instance definition 40.(ii) $\Phi(X)$, the last three predicates. These triplets return in the coefficient vector as a recurrent pattern "1 -1 -2". Registration in the m-th position results in a $\Phi(X)$ vector with m - 1 of these triplets, that all have their recurrent pattern in the coefficient vector. This way very larger theorem provers can be constructed in a conveniently systematic way.

4.4.4 The "Closed" Predicate and the Tree Closure

In this subparagraph definitions are given of perceptron schemes for closures and tree closures, preceded by a definition of a perceptron scheme for the hidden units needed to compute closures.

Definition 41 (of a perceptron scheme for hidden units in closures)

Let $\psi = \varphi_{c,i}$, index i is subject to generalisation of the first kind, index k to generalisation of the second kind;

$\Phi(X) = [\varphi_{fn,k,i}(X)\ \varphi_{fn,k,2i}(X)\ \varphi_{Th}(X)]$, $\mathbf{F}^+ = [1\ 1\ 1]$, and, by perceptron convergence procedure, $\mathbf{A} = [1\ 1\ -1]$.

Definition 42 (of a perceptron scheme for closures)

Let $\psi = \varphi_{cl,i}$; index i is subject to generalisation of the first kind, index j to generalisation of the second kind;

$\Phi(X) = [\varphi_{c,j}(X)]$, $\mathbf{F}^+ = [1]$, and by perceptron convergence procedure, $\mathbf{A} = [1]$.

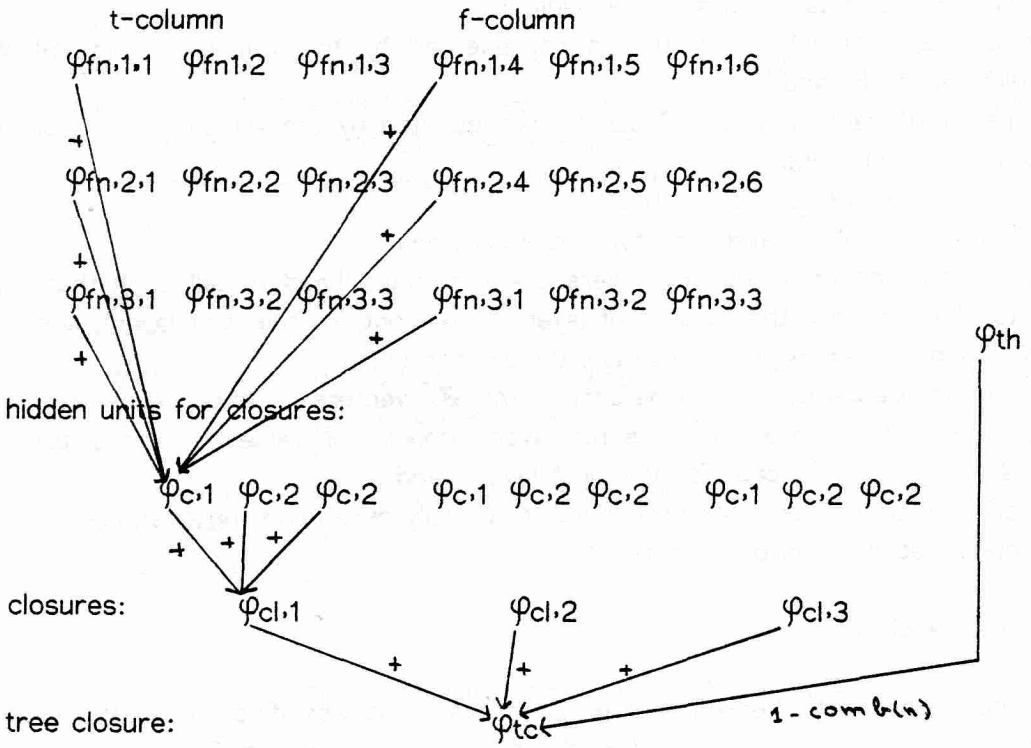
Definition 43 (of a perceptron scheme for tree closure)

Let $\psi = \varphi_{tc}$; index i is subject to generalisation of the second kind;

$\Phi(X) = [\varphi_{cl,i}(X)\ \varphi_{Th}(X)]$, $\mathbf{F}^+ = [1\ 1]$, and $\mathbf{A} = [1\ \ (1 - comb(n))]$.

Notice that in definition 43 the vector **A** has not been determined using the perceptron convergence procedure, this is impossible before n has been instantiated. The coefficient vector has been determined as follows: A tree structure comes with comb(n) fn's. There is a closure for each fn, which are represented by one φ_{cl} predicate each. φ_{tc} should have the value 1 if and only if for all φ_{cl}: $\varphi_{cl}(X) = 1$. Therefore an extra threshold of comb(n) - 1 should be added. Figure 16 illustrates the matter.

fig. 16.



4.5 The Operation of the Network is Correct

The operation of a network of perceptrons can be proved to conform to a certain specification if all perceptrons that make up the network can be garanteed to operate correctly. A Perceptron can be garanteed to operate correctly if it has a set of weights that has been determined using the perceptron convergence procedure, and that separates all $\mathbf{F}^+$ from all $\mathbf{F}^-$ vectors (see also the perceptron convergence theorem, paragraph 4.1).

The operation of a network of perceptrons can be said to conform to a certain specification if:

- there is a system of rules by which inputs and outputs of the network are represented by values of perceptron predicates; and
- there is a mapping of inputs onto outputs of the network (the specification of the operation of the network); and

- when the φ predicates of the first layer represent an input, the ψ predicates of the last layer represent an output of the network after evolution.

This is made precise in theorem 9.

Theorem 9:

Let NP be a network of n layers of perceptrons;

if:

- IN is a set of inputs for NP; and
- OUT is a set of outputs of NP; and
- TR : IN $\rightarrow$ OUT is a mapping that can be divided into n steps; and
- there is a representation system by which:
  - the elements of IN and OUT are represented by the values of perceptron predicates in NP, and
  - the result of each step of TR can be represented by the values of perceptron predicates in NP; and
- NP is divided into n layers such that:
  - perceptrons within a layer are not connected, and
  - the perceptrons of successive layers are connected head to tail such that the inputs of layer i are the outputs of layer i-1 (except for the first layer); and
- the coefficients of each perceptron are determined using:
  - the representation system to determine the $\mathbf{F}^+$ vectors of each perceptron, in such a way that the $\mathbf{F}^+$ vectors for layer i consist of values of the ψ predicates of layer i-1 (except for the first layer), and
  - the perceptron convergence procedure to actually determine the coefficients;

then NP computes n steps of TR correctly.

Proof: see appendix I.

The perceptron network theorem prover operates as is specified in virtue of the parallel tableau method, since it satisfies the conditions of theorem 9.

Chapter 5

Some Properties and Possible Extensions of the Theorem Prover

The size of a perceptron network in which the parallel tableau method is implemented:

The capacity of a tree structure is denoted with the letter n, the number of available atoms in an implementation is denoted by the letter p.

(a) the root:

n atom-variables are represented by:	np	predicates;
n unary-operator-variables are represented by:	n	predicates:
n-1 binary-operator-variables are represented by:	n-1	predicates;
the lpar- and rpar-variables are represented by: $n/2(n-1)-1$	$\sum_{i=2}^{n-1} i =$	predicates;

(b) "in-disjunction" predicates and final nodes:

comb(n+1) "in-disjunction" predicates require:	comb(n+1)	predicates;
each of the comb(n) fn's requires 2 n p predicates:	comb(n) 2 n p	predicates;
each fn requires $\sum_{i=1}^{n-1} i$ hidden units: comb(n) . $n/2(n-1)$.	comb(n)$\sum_{i=1}^{n-1} i =$	predicates;
each fn requires n + 1 predicates for a closure	comb(n) n + 1	predicates;
the tree closure requires:	1	predicate;
the threshold value predicate is:	1	predicate;

(c) in sum:

$comb(n+1) + comb(n)(2np + n + 1 + n/2(n-1)) + np + 2n + 1 + n/2(n-1) - 1 =$

$comb(n+1) + comb(n)(n^2/2 + 2np + n/2 + 1) + n^2/2 + 3n/2 + np =$

$comb(n+1) + (comb(n)(n^2 + n + 2 + 4np) + n^2 + 3n + 2np)/2.$

if n is odd, this can be simplified to:

$(comb(n)(n^2 + 4np + n + 6) + n^2 + 3n + 2np)/2.$

Growth of the network is exponential, due to the factor comb(n), compare 2^n with comb(n) and growth of the network (independent of p; p = 1):

n:	1	3	5	9	15	21	25
2^n:	2	8	32	512	32,768	2,097,152	33,554,432
comb(n):	1	3	10	126	6435	352,716	5,200,300
Network:	9	57	305	1671	8379	97,349,889	1,965,349,775

Let n be the capacity of a network representing a theorem prover. The number of layers of such a network is:

- one layer to represent a formula (input):	1
- one layer to compute the "in-disjunction" predicates:	1
- one layer for each tf-pair in a fn:	n
- one layer for each hidden unit for registration in a tf-pair:	n-1
- two layers to compute the closures	2
- one layer for the tree closure	1.

In sum: $2n + 4$

If the predicates of the network are updated in a synchronous fashion, the output of the network is computed in $2n + 3$ updatings of the neurons of the network. Since n is the capacity of a network, all formulas that can be analysed by the network are analysed in the same number of updatings. The number of updatings is a measure for the processing time of formulas, i.e. the time needed to analyse a formula. Growth of processing time is a linear function of n.

The theorem prover can be extended with a set DB of literals, which functions as a database against which the validity of formulas can be decided. To represent such a database in the perceptron network, only 2p predicates are needed, because:
if $AV = \{A,B,C\}$, then $\#(AV) = p = 3$, and

A is represented as	001
B is represented as	010
C is represented as	100
then A,B,C is represented as	111.

Since atoms need to be represented either as negated or not-negated 2p predicates are required. The database can be connected to a closure subnetwork to check it for inconsistency.

All fn's can be connected to the same database in the closure subnetworks. The advantage of this database is that it is very small. The disadvantage is that it does not contain complex formulas, so, for instance, neither disjunctions nor both the disjuncts of a disjunction, can be represented in the database.

Another extension is the implementation of the $\rightarrow$R semantic tableau reduction rule, and unlimited use of the $\sim$ connective.
$v_{\mathbb{I}}(\varphi \rightarrow \psi) = v_{\mathbb{I}}(\sim\varphi \vee \psi)$ (the formulas are logically equivalent)
which means that for $\chi = \varphi \rightarrow \psi$, both $D_{\chi}(\varphi,\psi)$ and $N_{\chi}(\varphi)$ hold. Apart from the fact that the representation of binary operators now takes two predicates, this can be implemented in two different ways:

(i) a feedback network, which changes the representation of $\varphi \rightarrow \psi$ into one of $\sim\varphi \vee \psi$, with registration as it is now. The drawback of this approach is that it might turn out to be impossible to divide the network up into perceptrons (due to the feedback). In that case not all connection weights can be determined by the perceptron convergence procedure, i.e. the network cannot be proven to operate as specified in virtue of the perceptron convergence theorem.

(ii) the predicate representing the $\rightarrow$ connective is connected to a network that inverts registration regarding negation, and is further interpreted as an v connective. Inverting registration with regard to negation is an XOR operation over the representations of the $\sim$ and $\rightarrow$ connectives.

It seems to me that (ii) is the best solution.

Unlimited use of the $\sim$ connective means that the theorem prover should be able to analyse formulas as $\sim(A \vee B)$ and $\sim\sim\sim\sim\sim B$. This requires both changes in the representation system for the root of a tree structure and the registration proces.

Formulas as the second example above make it necessary that symbols are represented in the root of a tree structure by arrays of predicates that can represent any symbol of $\mathbb{T}$, and since no symbol has a fixed place, room should be reserved in this arrays for parentheses with different scopes, as exemplified below.

Let $AV = \{A,B,C\}$, the capacity n of the tree structure of some implementation is 4, and $(^1$ is a parenthesis with a scope of one binary connective, then an array of nine predicates can represent all items:

$(^2$	$(^1$	$\sim$	v	C	B	A	$)^1$	$)^2$
$\varphi_{(,i}$	$\varphi_{(,j}$	$\varphi_{\sim,k}$	$\varphi_{\square,k}$	$\varphi_{r,k,1}$	$\varphi_{r,k,2}$	$\varphi_{r,k,3}$	$\varphi_{),m}$	$\varphi_{),h}$

arrays like this should replace all arrays of, and single, root variable representing predicates.

The registration proces should be extended to:

(i) finding out whether a formula is negated or not. A formula is negated if it has an odd number of $\sim$ connectives immediately left of it. A network to determine this is a network which determines whether an array of digits represents an odd or an even unary number (unary numbers, as in "binary numbers", "decimal numbers", etc.).

(ii) $v_{\mathbb{T}}(\sim(\varphi \vee \psi)) = v_{\mathbb{T}}(\sim\varphi \,\&\, \sim\psi)$, and $v_{\mathbb{T}}(\sim(\varphi \,\&\, \psi)) = v_{\mathbb{T}}(\sim\varphi \vee \sim\psi)$, (from De Morgans laws), so if a complex formula is negated, the value of unary, and binary-operator-variable representing predicates of its immediate constituents should be inverted. This is a recursive proces.

(iii) The $\varphi_{\square,i}$ predicates do not have a fixed place in the root any longer, so a network should be added which determines the value of i in each case an "in-disjunction" predicate should be evaluated. This network should connect certain $\varphi_{\square,i}$ and $\varphi_{(,j}$ predicates to the appropriate "in-disjunction" predicates, it will therefore probably be rather large, or fast growing with increasing capacities of implementations.

The size and growth of the theorem prover can be reduced if the theorem prover only needs to handle formulas that are conjunctions of disjunctions, or disjunctions of

conjunctions, e.g. $(\sim B \vee C \vee \sim D) \& A \& (F \vee \sim C)$, and $(\sim B \& C \& \sim D) \vee A \vee (F \& \sim C)$ respectively (this calls for redefinition of the language).

The possible reduction is a reduction of the number of "in-disjunction" perceptron predicates To determine whether two atomic occurrences are placed in disjunction it is now, due to the simpler form of the formulas, only necessary to find out whether a "v" connective occurs somewhere in between the occurrences under consideration. Since it is of no importance whether one or many v connectives occur between the ocurrences, one perceptron predicate per "in-disjunction" tableau predicate will do. This amounts to $\binom{n}{2}$ perceptron predicates, where n is the capacity of a tree structure. $\binom{n}{2} < 1/2\, n^2$, so this is a very nice reduction.

Appendix 1:

Theorems and their Proofs.

Theorem 3:

Let $\chi \in$ MFORM, φ and ψ are occurrences of atoms of $\mathbb{T}$; $D_\chi(\varphi,\psi)$ holds if and only if there are χ_1, χ_2 such that $\varphi \leq \chi_1$ and $\psi \leq \chi_2$, and in a full analysis of the sequent $\vdash \chi$, vR has been applied to either $\chi_1 \vee \chi_2$ or $\chi_2 \vee \chi_1$.

Proof:

"=>"

If $D_\chi(\varphi,\psi)$ holds, then, by definition 13, there are χ_1, χ_2 such that $\varphi \leq \chi_1$, $\psi \leq \chi_2$, and $\chi_1 \vee \chi_2 \leq \chi$ or $\chi_2 \vee \chi_1 \leq \chi$. In a full analysis of the sequent $\vdash \chi$ vR will therefore have been applied to either $\chi_1 \vee \chi_2$ or $\chi_2 \vee \chi_1$;

"<="

If $D_\chi(\varphi,\psi)$ does not hold, then, by definition 13, there are no χ_1, χ_2 such that $\varphi \leq \chi_1$ and $\psi \leq \chi_2$, and $\chi_1 \vee \chi_2 \leq \chi$ or $\chi_2 \vee \chi_1 \leq \chi$. In a full analysis of the sequent $\vdash \chi$ vR can therefore not have been applied to either $\chi_1 \vee \chi_2$ or $\chi_2 \vee \chi_1$.

Q.E.D.

Theorem 4:

Let $\chi \in$ MFORM, φ is an occurrence of an atom of $\mathbb{T}$; $N_\chi(\varphi)$ holds if and only if in a full analysis of the sequent $\vdash \chi$, $\sim$R has been applied to $\sim\varphi$.

Proof:

"=>"

If $N_\chi(\varphi)$ holds, then, by definition 15, $\sim\varphi \leq \chi$, so in a full analysis of the sequent $\vdash \chi$ $\sim$R has been applied to $\sim\varphi$.

"<="

If $N_\chi(\varphi)$ does not hold, then, by definition 15, $\sim\varphi$ is not a subformula of χ, so in a full analysis of the sequent $\vdash \chi$ $\sim$R cannot have been applied to $\sim\varphi$.

Q.E.D.

Theorem 5:

Let $\chi \in$ MFORM, φ,ψ are two occurrence of the same atom; $C_\chi(\varphi,\psi)$ holds if and only if in a full analysis of the sequent $\vdash \chi$, all simple sequents $\Pi \vdash \Gamma$ that contain both φ and ψ have the form ..,φ,..$\vdash$..,ψ,.. or ..,ψ,..$\vdash$..,φ,...

Proof:

"=>"

If $C_\chi(\varphi,\psi)$ holds, then, by definition 16:

(i) $D_\chi(\varphi,\psi)$ holds, so, by theorem 3 and the fact that only &R applications divide a

semantic tableau into two subtableaux, in a full analysis of the sequent $\vdash \chi$, $\Pi \vdash \Gamma$ contains occurrence φ if and only if it contains occurrence ψ; and

(ii) $N_\chi(\varphi)$ or $N_\chi(\psi)$ holds but not both, so, by theorem 4, $\sim$R has been applied to either $\sim\varphi$ or $\sim\psi$ in a full analysis of the sequent $\vdash \chi$, but not to both, therefore, by (i), $\varphi \in \Pi$ if and only if $\psi \in \Gamma$ and $\psi \in \Pi$ if and only if $\varphi \in \Gamma$;

therefore, by (i), (ii) and definition 15, in a full analysis of the sequent $\vdash \chi$ all simple sequents $\Pi \vdash \Gamma$ that contain both φ and ψ have the form ..,φ,..$\vdash$..,ψ,.. or ..,ψ,..$\vdash$..,φ,...

"<="

If $C_\chi(\varphi,\psi)$ does not hold, then, by definition 16, either:

(i) $D_\chi(\varphi,\psi)$ doesn't hold, so, by theorem 3 and the fact mentioned in the above case, if in a full analysis of the sequent $\vdash \chi$, $\Pi \vdash \Gamma$ contains either of the occurrences φ or ψ, but not both; or

(ii) $N_\chi(\varphi)$ and $N_\chi(\psi)$ both hold or both do not hold, so, by theorem 4, $\sim$R has been applied to either both $\sim\varphi$ and $\sim\psi$ or to not one of them in a full analysis of the sequent $\vdash \chi$, therefore, by (i), (a) $\varphi \in \Pi$ or $\psi \in \Gamma$, or (b) $\psi \in \Pi$ or $\varphi \in \Gamma$;

therefore, by (i), (ii), and definition 15, in a full analysis no simple sequent $\Pi \vdash \Gamma$ has the form ..,φ,..$\vdash$..,ψ,.. or ..,ψ,..$\vdash$..,φ,...

Q.E.D.

Lemma 3:

Let $\langle \chi$, PCES$\rangle$ be a PT; $\Pi \vdash \Gamma$ is a simple sequent in a full analysis of $\vdash \chi$ if and only if there is a PCE $\in$ PCES $\langle$TAT, FAT$\rangle$ such that Π = TAT and Γ = FAT.

Proof:

It will be proven that for all simple sequents $\Pi \vdash \Gamma$ in a full analysis of $\vdash \chi$ the ordered pair $\langle \Pi, \Gamma \rangle$ satisfies the definition of a PCE;

(i) for all atom occurrences $\varphi,\psi \leq \chi$: if $\varphi,\psi \in \Pi \cup \Gamma$ then, &R cannot have been applied to the principal operator connecting φ and ψ, so it had to be $\vee$R, thus, by theorem 3, $D_\chi(\varphi,\psi)$ holds;

(ii) $\Pi \cup \Gamma$ is complete concerning (i) because the supposition that $D_\chi(\varphi',\varphi)$ holds while $\varphi \in \Pi \cup \Gamma$ and $\varphi' \notin \Pi \cup \Gamma$ contradicts theorem 3, since, following the supposition, &R had to be applied to the principal operator connecting φ and φ';

(iii) for all $\varphi \in \Pi$: $\sim$R has been applied to $\sim\varphi$, so, by theorem 4, $N_\chi(\varphi)$ holds;

(iv) for all $\varphi \in \Gamma$: $\sim$R has not been applied to $\sim\varphi$, so, by theorem 4, $N_\chi(\varphi)$ doesn't hold;

from (i), (ii), (iii), and (iv) it follows that

(a) for each simple sequent $\Pi \vdash \Gamma$ in a full analysis of $\vdash \chi$, there is, by definition 18, a PCE $\in$ PCES $\langle$TAT, FAT$\rangle$ of $\langle \chi$, PCES$\rangle$ such that Π = TAT and Γ = FAT;

(b) for all PCE's $\in$ PCES of $\langle \chi$, PCES$\rangle$ there is, by theorem 1, a simple sequent $\Pi \vdash \Gamma$ in a full analyis of $\vdash \chi$ such that Π = TAT and Γ = FAT.

Q.E.D.

Lemma 4:

if $\langle \varphi$, PCES$\rangle$ is an open PT, then φ is invalid and there is at least one PCE $\in$ PCES $\langle$TAT, FAT$\rangle$ which is open, and an interpretation of TAT U FAT such that:

$\mathbf{I}(\varphi) = \mathbf{t}$ for all $\psi \in$ TAT; and
$\mathbf{I}(\varphi) = \mathbf{f}$ for all $\psi \in$ FAT,
is a counter-example for φ.

Proof:

if $\langle \varphi$, PCES$\rangle$ is an open PT, then, by definition 19, there is a PCE $\in$ PCES which is open; if this PCE = $\langle$TAT, FAT$\rangle$ then there is, by lemma 3, a simple sequent $\Pi \vdash \Gamma$ in a full analysis of $\vdash \varphi$, such that Π = TAT and Γ = FAT, which, by theorem 5, is open; by lemma 2, the sequent $\Pi \vdash \Gamma$ is incorrect and has a counter-example; by lemma 1: a counter-example for $\Pi \vdash \Gamma$ is a counter-example for $\vdash \varphi$; a counter-example for $\vdash \varphi$ is a counter-example for φ; if φ has a counter-example, it is invalid, by definition 6.

Q.E.D.

Theorem 6 (completeness theorem):

Let $\varphi \in$ MFORM; φ is valid if and only if a PT $\langle \varphi$, PCES$\rangle$ is closed.

Proof:

"=>"

if φ is valid, then, by definition 6, φ has no counter-examples; if φ does not have any counter-examples, then $\vdash \varphi$ doesn't have any either; by lemma 2, $\vdash \varphi$ is correct; if $\vdash \varphi$ is correct then, by theorem 1, a full analysis of the sequent $\vdash \varphi$ is closed; by definition 8.b, all simple sequents $\Pi \vdash \Gamma$ in a full analysis of the sequent $\vdash \varphi$ are closed; by lemma 3, for all simple sequents $\Pi \vdash \Gamma$ in a full analysis of the sequent $\vdash \varphi$, and all PCE $\in$ PCES $\langle$TAT, FAT$\rangle$, if Π = TAT and Γ = FAT then $\langle$TAT, FAT$\rangle$ is closed; by definition 19.a, $\langle \varphi$, PCES$\rangle$ is closed;

"<="

see lemma 4.

Q.E.D.

Theorem 7:

Let $\langle \varphi$, PCES$\rangle$ be a PT, t = $\langle$root, final nodes$\rangle$ is a tree structure; if R(root) = φ, then for each PCE $\in$ PCES, $\langle$TAT, FAT$\rangle$, there is a fn_i in final nodes and an $occurs_i$ in $OCCURS_t$ such that R(REG(root, fn_i, $occurs_i$)) = $\langle$TAT, FAT$\rangle$.

Proof:

let R(root) = φ; in virtue of the Ordering Method: if two PCE's $\in$ PCES $\langle TAT_i, FAT_i\rangle$ and $\langle TAT_j, FAT_j\rangle$ can only be registered in one fn, using the same registration recipe, then if $TAT_i \cup FAT_i$ = UI and $TAT_j \cup FAT_j$ = UJ, then UI $\subset$ UJ or UJ $\subset$ UI.
But then either $\langle TAT_i, FAT_i\rangle$ or $\langle TAT_j, FAT_j\rangle$ is not a PCE.

Let UI={A,B}, and UJ={A,B,...};

(i) Suppose an interpretation for UI is a counter-example for the sequent $\vdash \varphi$ i.e. $\langle TAT_i, FAT_i\rangle$ is a PCE, then $\langle TAT_j, FAT_j\rangle$ cannot be a PCE, because A and B are not placed in disjunction with any other atom occurrence in φ, otherwise such an occurrence would be an element of UI, so $\langle TAT_j, FAT_j\rangle$ contains atom occurren-

ces that are not placed in disjunction and is therefore not a PCE;
if $\langle TAT_j, FAT_j \rangle$ is a PCE, then $\langle TAT_i, FAT_i \rangle$ cannot be a PCE, because A and B are both placed in disjunction with other occurrences of atomic subformulas of φ, occurrences that are elements of UJ but not of UI, so UI is not complete concerning disjuncts. Therefore $\langle TAT_i, FAT_i \rangle$ is not a PCE.

From (i) and (ii) it follows that there can be no two PCE's for which there is only one fn and only one registration recipe that generates both PCE's.

Q.E.D.

Theorem 9:

Let NP be a network of n layers of perceptrons;
if:

- IN is a set of inputs for NP; and
- OUT is a set of outputs of NP; and
- TR : IN $\rightarrow$ OUT is a mapping that can be divided into n steps; and
- there is a representation system by which:
  - the elements of IN and OUT are represented by the values of perceptron predicates in NP, and
  - the result of each step of TR can be represented by the values of perceptron predicates in NP; and
- NP is divided into n layers such that:
  - perceptrons within a layer are not connected, and
  - the perceptrons of successive layers are connected head to tail such that the inputs of layer i are the outputs of layer i-1 (except for the first layer); and
- the coefficients of each perceptron are determined using:
  - the representation system to determine the $\mathbf{F}^+$ vectors of each perceptron, in such a way that the $\mathbf{F}^+$ vectors for layer i consist of values of the ψ predicates of layer i-1 (except for the first layer), and
  - the perceptron convergence procedure to actually determine the coefficients;

then NP computes n steps of TR correctly.

Proof (by induction to the number of steps in T):

(1) NP computes one step of TR correctly, because:
  - in NP all perceptrons operate correctly (by perceptron convergence procedure);
  - in each layer of NP the perceptrons operate independently of one another; so each layer of NP computes one step of TR correctly;

(2) induction hypothesis: suppose the NP computes m steps of TR correctly; since:
  - the inputs of the m + 1-th layer of NP are represented by the predicates that represent the outputs of the m-th layer of NP, which are computed correctly (by induction hypothesis); and
  - the m + 1-th layer of NP computes one step of TR correctly (by (1));

  NP computes m + 1 steps of TR correctly;

(3) according to the induction principle this means that NP computes n steps of TR correctly.

Q.E.D.

Appendix 2:

BNF Notation

BNF notation is a way to define the structure of formal expressions. BNF stands for Backus Naur Form, Backus and Naur are computer scientists.

The symbols of the BNF notation are: ::=, |, <, >, [,], {, }.

In BNF notation non-terminal symbols, symbols between "<" and ">", are (possibly in an indirect way) defined by terminal symbols, symbols that are neither BNF symbols nor enclosed by hooks.

<A> ::= M means that A is defined as M;

<A> ::= <M> <M> ::= B, is an indirect definition of A as B

"|" denotes an alternative choise, i.e. <A> ::= B | C means that A is defined to be either B or C;

Items between [ ] may occur nil or one times, <A> ::= B [C] tells us that both B and B C satisfy the definition of A;

Items between { } may occur nil or more times, <A> ::= B {D C} tells us that both B and B D C, B D C D C, and B D C D C D C, etc. satisfy the definition of A;

Several points may be used together with "|", "[ ]", and { } to indicate an infinity, if it is clear what could be meant, for instance in <n> ::= 0 | 1 | 2 | 3 . . . , n is defined to be any of the natural numbers.

Appendix 3:

C-Listing of a Perceptron Network
Theorem Prover Simulation Program

```
/********************** neuralog.c ******************************/

/*                        By Marc Drossaers                      */

/************************** includes ****************************/
     #include <stdio.h>
     #include <string.h>

/************************** globals *****************************/
char readstring [45];
int root [75];
int D_preds[35];
int neg_plces[]= {4,16,28,40,52,64};
int op_plces[]= {11,24,37,50,63};
int atomplces[]= {5,17,29,41,53,65};
int op_i, at_i, neg_i;
int f_nodes[20][6][12];
int th_value=1;
int labels[20] [6]= {    {1,2,3,4,5,6},
                         {2,3,4,5,6,1},
                         {3,4,5,6,1,2},
                         {4,5,6,1,2,3},
                         {5,6,1,2,3,4},
                         {6,1,2,3,4,5},
                         {1,3,4,6,5,2},
                         {2,4,5,1,6,3},
                         {3,5,6,2,1,4},
                         {4,6,1,2,3,5},
                         {5,1,2,3,4,6},
                         {6,2,3,4,5,1},
                         {1,4,5,6,2,3},
                         {2,5,6,4,3,1},
                         {3,6,1,5,4,2},
                         {4,1,2,5,6,3},
                         {5,2,3,6,1,4},
                         {6,3,4,1,2,5},
                         {1,5,3,4,6,2},
                         {2,6,4,3,1,5}  };
int CLS_preds[20][6];
int hu_validity[20];
int valid;
/************************** functions ***************************/
void initialise (void);
void read_form (void);
int pars_form (void);
int once_more (void);
void do_par(int kind, int offset);
int do_atom (int kind);
void do_operator (int kind);
```

```
void do_negation (int offset);
void print_root (void);
void in_disjunction (void);
void print_D_preds (void);
void registration(void);
void regist_top(int node);
int SUM_D(int pos, int node);
void regist_rest(int node,int pos);
void print_f_nodes(void);
int present_occ(int node, int pos);
int get_hu(int present, int occ1, int occ2);
void closure (void);
void validity (void);
void print_validity(void);
void print_all (void);
/************************* main *********************************/

main ()
{
     do
     {    do
          {    initialise();
               read_form();
          } while (pars_form());
          in_disjunction();
          registration();
          closure();
          print_all();
     } while (once_more());

     return 0;
}

/********************** general functions ***********************/
void initialise (void)
{
     register int i,j,k;

     for (i=0; i<75; root[i++]=0);
     op_i=neg_i=at_i=0;
     for (i=0; i<20; i++)
     for (j=0; j<6; j++)
     for (k=0; k<12; k++)
          f_nodes[i][j][k]=CLS_preds[i][j]=hu_validity[i]=0;
}

int once_more (void)
{    char yn;

     printf("\n%48s","ONCE MORE? [Y/N] ");
     scanf("%s", &yn);
     return (yn=='y' || yn=='Y') ? 1 : 0;
}

/******************** read a formula ****************************/
```

```
void read_form (void)
{
 printf("%23s", "ENTER A FORMULA, MAX. LENGTH IS SIX ATOMS");

 printf("%21s", "THE SET OF ATOMS IS {A, B, C, D, E, F}");
 printf("\n%25s", " ");
 scanf("%s", readstring);
}

/***************** pars and encode the formula ******************/

int pars_form (void)
{    int atom=5, i=0, j=0, k=0, m, n, stack [14][2], scope;

          n=strlen(readstring);
        while (i<n)
        { m=(int) readstring[i];
          if (m=='(')
          {    stack[k][0]=j;
                    stack[k][1]=atom;
                    k++;
          } else
          if (m>='A' && m<='F') {j++; atom=do_atom(m);} else
          if (m=='&' || m=='v') do_operator(m); else
          if (m=='~') do_negation(atom); else
          if (m==')')
          {   k--;
              scope=(j-stack[k][0])-1;
              root[stack[k][1]-(scope+1)]=1;
              root[atomplces[j-1]+(scope+5)]=1;
          }
          else { printf("\n%52s", "NOT A WELL-FORMED FORMULA");
                 printf("\n\n%50s", "HIT RETURN TO RESTART");
                 getchar();
                 getchar();
                 return 1;
                }
          i++;
        }

        printf("\n");
        return 0;
}

int do_atom(int kind)
{    int n;

        n=atomplces[at_i];
        switch (kind)
        { case 'B': n++; break;
          case 'C': n+=2; break;
          case 'D': n+=3; break;
          case 'E': n+=4; break;
          case 'F': n+=5; break;
        }
        root[n]=1;
```

```
        at_i++;
       return atomplces[at_i];
}

void do_operator (int kind)
{    int n;


     if (kind=='v') root[n=op_plces[op_i],n]=1;
     op_i++;

}

void do_negation (int offset)
{
     root[offset-1]=1;
}

void print_root (void)
{
int  grouping []=
       {4,1,1,1,4,1,1,1,1,3,1,1,2,1,2,1,1,3,1,1,1,1,4,1,1,1,4};
int i=0,j,k,l,m=0,n;

printf("root predicates:\n\n");

     for (l=0; l<27; l++)
     {    k=grouping[l];
          for  (j=0; j<k; j++)
          {    if (m==atomplces[i]+1) {m+=5; i++;}
               if (m==0) printf("%5d",root[m]);
               else printf("%d", root[m]);
               m++;
          }
          printf(" ");
     }

     printf("\n");
     for (l=1; l<6; l++)
     {    for (k=0; k<6; k++)
          {    j=atomplces[k]+1;
               if (k==0) printf("%12d", root[j]); else
               if (k==1 || k==5) printf("%11d",root[j]);
               else printf("%12d", root[j]);
          }
          printf("\n");
     }
}

/*************** in-disjunction predicates *********************/

void in_disjunction (void)
{
D_preds[0]=root[11];                                    /*D(1,2)*/
D_preds[1]=(root[11]>root[3]);                         /*D(1,3),1*/
```

```
D_preds[2]=(root[24]>(root[15]+root[14]+root[13]+
                                        root[12])) /*D(1,3),2*/
D_preds[3]=(root[11]>(root[3]+root[2]));           /*D(1,4),1*/
D_preds[4]=(root[24]>(root[2]+root[14]+root[15]+root[13]+
           root[12]));                             /*D(1,4),2*/
D_preds[5]=(root[37]>(root[27]+root[26]+root[25]+root[14]+
           root[13]+root[12]));                    /*D(1,4),3*/
D_preds[6]=(root[11]>(root[3]+root[2]+root[1]));   /*D(1,5),1*/
D_preds[7]=(root[24]>(root[2]+root[1]+root[15]+root[14]+
           root[13]+root[12]));                    /*D(1,5),2*/
D_preds[8]=(root[37]>(root[1]+root[14]+root[13]+root[12]+
           root[27]+root[26]+root[25]));           /*D(1,5),3*/
D_preds[9]=(root[50]>(root[13]+root[12]+root[26]+root[25]+
           root[39]+root[38]));                    /*D(1,5).4*/
D_preds[10]=(root[11]>(root[3]+root[2]+root[1]+
                                    root[0]));     /*D(1,6),1*/
D_preds[11]=(root[24]>(root[2]+root[1]+root[0]+root[15]+
            root[14]+root[13]+root[12]));          /*D(1,6),2*/
D_preds[12]=(root[37]>(root[1]+root[0]+root[14]+root[13]+
            root[12]+root[27]+root[26]+root[25])); /*D(1,6),3*/
D_preds[13]=(root[50]>(root[0]+root[13]+root[12]+root[26]+
            root[25]+root[39]+root[38]));          /*D(1,6),4*/
D_preds[14]=(root[63]>(root[12]+root[25]+root[38]+
                                        root[51])); /*D(1,6),5*/
D_preds[15]=root[24];                                /*D(2,3)*/
D_preds[16]=(root[24]>root[15]);                   /*D(2,4),1*/
D_preds[17]=(root[37]>(root[27]+root[26]+root[25]));/*D(2,4),2*/
D_preds[18]=(root[24]>(root[14]+root[15]));        /*D(2,5),1*/
D_preds[19]=(root[37]>(root[14]+root[25]+root[26]+
                                        root[27])); /*D(2,5),2*/
D_preds[20]=(root[50]>(root[26]+root[25]+root[39]+
                                        root[38])); /*D(2,5),3*/
D_preds[21]=(root[24]>(root[13]+root[14]+root[15]));/*D(2,6),1*/
D_preds[22]=(root[37]>(root[13]+root[14]+root[25]+ root[26]+
            root[27]));                            /*D(2,6),2*/
D_preds[23]=(root[50]>(root[13]+root[25]+root[26]+root[38]+
            root[39]));                            /*D(2,6),3*/
D_preds[24]=(root[63]>(root[25]+root[38]+root[51]));/*D(2,6),4*/
D_preds[25]=root[37];                              /*D(3,4)  */
D_preds[26]=(root[37]>root[27]);                   /*D(3,5),1*/
D_preds[27]=(root[50]>(root[39]+root[38]));        /*D(3,5),2*/
D_preds[28]=(root[37]>(root[26]+root[27]));        /*D(3,6),1*/
D_preds[29]=(root[50]>(root[26]+root[38]+root[39]));/*D(3,6),2*/
D_preds[30]=(root[63]>(root[38]+root[51]));        /*D(3,6),3*/
D_preds[31]=root[50];                              /*D(4,5)  */
D_preds[32]=(root[50]>root[39]);                   /*D(4,6),1*/
D_preds[33]=(root[63]>root[51]);                   /*D(4,6),2*/
D_preds[34]=root[63];                              /*D(5,6)  */
}

void print_D_preds (void)
{int i,j,k,l=0,m;

     printf("\n");
```

```
     for (i=0; i<5; i++)
          for (j=i+1; j<+6; j++)
            {    for (k=i; k<j; k++)
             {
              if (k==i && i==0) printf("%14d", D_preds[l]);else
              if (k==i && i==1) printf("%27d", D_preds[l]);else
              if (k==i && i==2) printf("%40d", D_preds[l]);else
              if (k==i && i==3) printf("%53d", D_preds[l]);else
              if (k==i && i==4) printf("%66d", D_preds[l]);
              else printf("%13d", D_preds[l]);
              l++;
             }
              printf("\n");
           }
}

/************************ registration  ***************************/

void registration(void)
{ int pos, node;

     for (node=0; node<20; node++)
     {    regist_top(node);
          for (pos=1; pos<6; pos++) regist_rest(node,pos);
     }
}

void regist_top(int node)
{int k,l,occ, occ_ad, neg_ad;

 occ=labels[node][0];
 occ_ad=atomplces[occ-1];
 neg_ad=occ_ad-1;

 for (k=0; k<6; k++)
 f_nodes[node][0][k]=((root[occ_ad+k]+root[neg_ad]-th_value)>0);
 for (k=6; k<12; k++)
 f_nodes[node][0][k]=((root[occ_ad+(k-6)]-root[neg_ad])>0);
}

void regist_rest(int node, int pos)
{    int k,l,
     occ, occ_ad, neg_ad,
     atom, negation, same,
     present []= {0,0,0,0,0}, SUM_present=0,
     SUM_hu=0;

     occ=labels[node][pos];
     occ_ad=atomplces[occ-1];
     neg_ad=occ_ad-1;

     for (k=0; k<pos; k++)
     {    present[k]=present_occ(node, k);
          SUM_hu+=get_hu(present[k], labels[node][k], occ);
          SUM_present+=(present[k]<100);
     }
```

```
      for (k=0; k<6; k++)
      {     same=0;
            for (l=0; l<pos; l++) same+=f_nodes[node][l][k];
            atom=root[occ_ad+k];
            negation=root[neg_ad];

            f_nodes[node][pos][k]=
              ((negation+atom+SUM_hu-SUM_present-same-th_value)>0);
      }
      for (k=6; k<12; k++)
      {     same=0;
            for (l=0; l<pos; l++) same+=f_nodes[node][l][k];

            atom=root[occ_ad+(k-6)];
            negation=root[neg_ad];
            f_nodes[node][pos][k]=
                        ((atom-negation+SUM_hu-SUM_present-same)>0);
      }
}

int present_occ(int node, int pos)
{ int k;

      for(k=0; k<12; k++) if (f_nodes[node][pos][k]) return k;
      return 100;
}

int get_hu(int present, int occ1, int occ2)
{     int in_disj;

      if (occ1<occ2) in_disj=SUM_D(occ1,occ2);
      else in_disj=SUM_D(occ2,occ1);

      return (((present<100)+in_disj-th_value)>0);
}

void print_f_nodes(void)
{     int i,j,k;
      char letterT, letterF;

      printf("\n");
      for (i=0; i<20; i++) printf(" %s%s ", "T", "F");

      for (j=0; j<6; j++)
       for (i=0; i<20; i++)
            {     letterT=letterF=32;
                        for (k=0; k<12; k++)
                  {     if (f_nodes[i][j][k]==1)
                              if (k<6) letterT=65+k;
                              else letterF=65+(k-6);
                  }
                  printf("%d%c%c ", labels[i][j], letterT, letterF);

            }
}
```

```
int SUM_D(int i, int j)
{return i==1 && j==2 ? D_preds[0]>0 :
   i==1 && j==3 ? D_preds[1]+D_preds[2]>0 :
   i==1 && j==4 ? D_preds[3]+D_preds[4]+D_preds[5]>0 :
   i==1 && j==5 ? D_preds[6]+D_preds[7]+D_preds[8]+D_preds[9]:
   i==1 && j==6 ? D_preds[10]+D_preds[11]+D_preds[12]+

                                          D_preds[13]+D_preds[14]>0 :
   i==2 && j==3 ? D_preds[15]>0 :
   i==2 && j==4 ? D_preds[16]+D_preds[17]>0 :
   i==2 && j==5 ? D_preds[18]+D_preds[19]+D_preds[20]>0 :
   i==2 && j==6 ? D_preds[21]+D_preds[22]+D_preds[23]+
                                                 D_preds[24]>0 :
   i==3 && j==4 ? D_preds[25]>0 :
   i==3 && j==5 ? D_preds[26]+D_preds[27]>0 :
   i==3 && j==6 ? D_preds[28]+D_preds[29]+D_preds[30]>0 :
   i==4 && j==5 ? D_preds[31]>0 :
   i==4 && j==6 ? D_preds[32]+D_preds[33]>0 : D_preds[34]>0;

}

/*************************** closure ***************************/

void closure(void)
{    int node, pos, pred,
     sumT, sumF;

     for (node=0; node<20; node++)
     for (pred=0; pred<6; pred++)
     {    sumT=sumF=0;
          for (pos=0; pos<6; pos++)
          {    sumT+=f_nodes[node][pos][pred];
               sumF+=f_nodes[node][pos][pred+6];
          }
               CLS_preds[node][pred]=((sumT+sumF-th_value)>0);
     }
     validity();
}

void validity (void)
{ int node, pred, SUM_valid=0, SUM_cls;

     for (node=0; node<20; node++)
     {    SUM_cls=0;
          for (pred=0; pred<6; pred++)
              SUM_cls+=CLS_preds[node][pred];
          hu_validity[node]=(SUM_cls>0);
          SUM_valid+=hu_validity[node];
     }
     valid=((SUM_valid-(19*th_value))>0);
}

void print_validity(void)
{    int node;
     printf("\n");
```

```
      for (node=0; node<20; node++)
         printf("  %d ", hu_validity[node]);

     if (valid) printf("\n%50s", "THE FORMULA IS VALID\n");
     else printf("\n%51s","THE FORMULA IS INVALID\n");
}


void print_all(void)
{    print_root();
     printf("\n%51s","HIT RETURN TO CONTINUE");
     getchar();
     getchar();
     printf("in-disjunction predicates:\n"); print_D_preds();
     printf("\n%51s","HIT RETURN TO CONTINUE");getchar();

     printf("final nodes:\n"); print_f_nodes();
     printf("\n%51s","HIT RETURN TO CONTINUE");getchar();
     printf("closure of final nodes:\n"); print_validity();
}
```

Literature on the subject

Ackley, D.H, Hinton, G.E. & Sejnowski, T.J. (1985) *A Learning Algorithm for Boltzmann Machines,*
in: Cognitive Science, 9, 147 - 169.
Amit, D.J. (1987) *Neural Networks - Achievements, Prospects, Difficulties,*
in: The Physics of Structure Formation, Int. symposium, Tubingen.
Amit, D.J, Gutfreund, H. & Sompolinski, H. (1987) *Information Storage in Neural Networks with Low Levels of Activity,*
in: A Physical Review, 1, 2293 - 2303.
Amit, D.J, Gutfreund, H. & Sompolinski, H. (1985) *Spin-glass Models of Neural Networks,*
in: A Physical Review, vol 32, 2, 1007 - 1018.
Anderson, J.A. & Hinton, G.E. (1981) *Models of Information Processing in the Brain,*
in: **Hinton, G.E. & Anderson, J.A. (eds.)** (1981) *Parallel Models of Associative Memory,*
Hillsdale New Jersey: Erlbaum.
Ballard, D.H. (1986) *Cortical Connections and Parallel Processing: Structure and Function,*
in: The Behavioural and Brain Sciences, 6, 67 - 120.
Ballard, D.H. (1987) *Parallel Logical Inference and Energy Minimization,*
Report TR142, Computer Science Department, University of Rochester.
Ballard, D.H. & Hayes, P.J. (1984) *Parallel Logical Inference,*
Presented at the 7th Annual Conf. of the Cognitive Science Society, Boulder Colorado.
Barth, E.M. & Krabbe, E.C.W. (1982) *From Axiom to Dialogue: a Philosophical Study of Logics and Argumentation,*
Berlin, New York: Walter de Gruyter.
Coolen, A.C.C (1989) *Inleiding Neurale Netwerken,*
Syllabus.
Dorf, R.C. (1969) *Matrix Algebra: a Programmed Introduction,*
New York: Wiley.
Feldman, J.A. & Ballard, D.H. (1982) *Connectionist Models and Their Properties,*
Cognitive Science, 6, 205 - 254.
Hebb, D.O. (1949) *The organization of Behaviour,*
New York: Wiley.
Hillis, W.D. (1985) *The Connection Machine,*
Cambridge Mass: MIT Press.
Hodges, W.
(1977 1982) *Logic,* Harmondsworth, England: Penguin Books.
Hopfield, J.J. (1982) *Neural Networks and Physical systems with emergent collective computational abilities,*
in: Proc. Natl. Acad. Sci. USA, 79, 2554 - 2558.
Hopfield, J.J. & Tank, D.W. (1985) *"Neural" Computation of Decisions in Optimization Problems,*
in: Biol. Cybern, 52, 141 - 152.

Klop, J.W. & Meyer, J.-J.Ch. (1987) *Toegepaste Logica: Deel 1 Resolutielogica,*
Syllabus.
Kohonen, T. (1977) *Associative Memory: A System Theoretical Approach,*
New York: Springer.
Kohonen, T. (1988) *Self-Organisation and Associative Memory,*
New York: Springer.
Kosko, B. (1987) *Bidirectional Associative Memories,*
in: IEEE transactions on Systems, Man, and Cybernetics.
Little, W.A. (1974) *The Existence of Persistent states in the Brain*
in: Math. Biosc, 19, 101 - 120.
Lloyd, J.W. (1984) *Foundations of Logic Programming,*
Berlin: Springer.
McCulloch, W.S. & Pitts, W.H. (1943) *A Logical Calculus of the Ideas Immanent in Nervous Activity,*
in: Bull. Math. Biophys, 5, 115 - 135.
Minsky, M.L. & Papert, S.A. (1969, 1988) *Perceptrons,*
Cambridge Mass: MIT Press.
Mozer, M.C. (1987) *Rambot, a Connectionist Expert System that learns by Example,*
in: Proc. IEEE First Anual Conf. on Neural Networks, San Diego.
Rosenblatt, F. (1962) *Principles of Neurodynamics,*
New York: Spartan.
Rumelhart, D.E, McClelland, J.L. & the PDP Research Group. (1986) *Parallel Distributed Processing,*
Cambridge Mass: MIT Press.
Sompolinsky, H. & Kanter, I. (1986) *Temporal Association in Asymmetric Neural Networks,*
in: Physical Review Letters, vol 57, 2861 - 2864.

Logic Group Preprint Series

Department of Philosophy
University of Utrecht
Heidelberglaan 2
3584 CS Utrecht
The Netherlands

1 C.P.J. Koymans, J.L.M. Vrancken, *Extending Process Algebra with the empty process*, September 1985
2 J.A. Bergstra, *A process creation mechanism in Process Algebra*, September 1985
3 J.A. Bergstra, *Put and get, primitives for synchronous unreliable message passing*, October 1985
4 A. Visser, *Evaluation, provably deductive equivalence in Heyting's arithmetic of substitution instances of propositional formulas*, November 1985
5 G.R. Renardel de Lavalette, *Interpolation in a fragment of intuitionistic propositional logic*, January 1986
6 C.P.J. Koymans, J.C. Mulder, *A modular approach to protocol verification using Process Algebra*, April 1986
7 D. van Dalen, F.J. de Vries, *Intuitionistic free abelian groups*, April 1986
8 F. Voorbraak, *A simplification of the completeness proofs for Guaspari and Solovay's R*, May 1986
9 H.B.M. Jonkers, C.P.J. Koymans & G.R. Renardel de Lavalette, *A semantic framework for the COLD-family of languages*, May 1986
10 G.R. Renardel de Lavalette, *Strictheidsanalyse*, May 1986
11 A. Visser, *Kunnen wij elke machine verslaan? Beschouwingen rondom Lucas' argument*, July 1986
12 E.C.W. Krabbe, *Naess's dichotomy of tenability and relevance*, June 1986
13 H. van Ditmarsch, *Abstractie in wiskunde, expertsystemen en argumentatie*, Augustus 1986
14 A. Visser, *Peano's Smart Children, a provability logical study of systems with built-in consistency* , October 1986
15 G.R. Renardel de Lavalette, *Interpolation in natural fragments of intuitionistic propositional logic*, October 1986
16 J.A. Bergstra, *Module Algebra for relational specifications*, November 1986
17 F.P.J.M. Voorbraak, *Tensed Intuitionistic Logic*, January 1987
18 J.A. Bergstra, J. Tiuryn, *Process Algebra semantics for queues*, January 1987
19 F.J. de Vries, *A functional program for the fast Fourier transform*, March 1987
20 A. Visser, *A course in bimodal provability logic*, May 1987
21 F.P.J.M. Voorbraak, *The logic of actual obligation, an alternative approach to deontic logic*, May 1987
22 E.C.W. Krabbe, *Creative reasoning in formal discussion*, June 1987
23 F.J. de Vries, *A functional program for Gaussian elimination*, September 1987
24 G.R. Renardel de Lavalette, *Interpolation in fragments of intuitionistic propositional logic*, October 1987 (revised version of no. 15)
25 F.J. de Vries, *Applications of constructive logic to sheaf constructions in toposes*, October 1987
26 F.P.J.M. Voorbraak, *Redeneren met onzekerheid in expertsystemen*, November 1987
27 P.H. Rodenburg, D.J. Hoekzema, *Specification of the fast Fourier transform algorithm as a term rewriting system*, December 1987

28 D. van Dalen, *The war of the frogs and the mice, or the crisis of the Mathematische Annalen,* December 1987
29 A. Visser, *Preliminary Notes on Interpretability Logic,* January 1988
30 D.J. Hoekzema, P.H. Rodenburg, *Gauß elimination as a term rewriting system*, January 1988
31 C. Smoryński, *Hilbert's Programme,* January 1988
32 G.R. Renardel de Lavalette, *Modularisation, Parameterisation, Interpolation,* January 1988
33 G.R. Renardel de Lavalette, *Strictness analysis for POLYREC, a language with polymorphic and recursive types*, March 1988
34 A. Visser, *A Descending Hierarchy of Reflection Principles,* April 1988
35 F.P.J.M. Voorbraak, *A computationally efficient approximation of Dempster-Shafer theory*, April 1988
36 C. Smoryński, *Arithmetic Analogues of McAloon's Unique Rosser Sentences,* April 1988
37 P.H. Rodenburg, F.J. van der Linden, *Manufacturing a cartesian closed category with exactly two objects*, May 1988
38 P.H. Rodenburg, J.L.M.Vrancken, *Parallel object-oriented term rewriting : The Booleans*, July 1988
39 D. de Jongh, L. Hendriks, G.R. Renardel de Lavalette, *Computations in fragments of intuitionistic propositional logic*, July 1988
40 A. Visser, *Interpretability Logic,* September 1988
41 M. Doorman, *The existence property in the presence of function symbols,* October 1988
42 F. Voorbraak, *On the justification of Dempster's rule of combination*, December 1988
43 A. Visser, *An inside view of EXP, or: The closed fragment of the provability logic of* $I\Delta_0+\Omega_1$, February 1989
44 D.H.J. de Jongh & A. Visser, *Explicit Fixed Points in Interpretability Logic,* March 1989
45 S. van Denneheuvel & G.R. Renardel de Lavalette, *Normalisation of database expressions involving calculations*, March 1989
46 M.F.J. Drossaers, *A Perceptron Network Theorem Prover for the Propositional Calculus*, July 1989