Report Number: WUCS-93-10

1993-01-01

A Fault Tolerant Connectionist Architecture for Construction of Logic Proofs

Gadi Pinkas

This chapter considers the problems of expressing logic and constructing proofs in fault tolerant connectionist networks that are based on energy minimalism. Given a first-order-logic knowledge base and a bound k, a symmetric network is constructed (like a Boltzman machine or a Hopfield network) that searches for a proof for a given query. If a resolution-based proof of length no longer than k exists, then the global minima of the energy function that is associated with the network represent such proofs. If no proof exist then the global minima indicate the lack of a proof. The network that is... **Read complete abstract on page 2.**

A Fault Tolerant Connectionist Architecture for Construction of Logic Proofs

Gadi Pinkas

Complete Abstract:

This chapter considers the problems of expressing logic and constructing proofs in fault tolerant connectionist networks that are based on energy minimalism. Given a first-order-logic knowledge base and a bound k, a symmetric network is constructed (like a Boltzman machine or a Hopfield network) that searches for a proof for a given query. If a resolution-based proof of length no longer than k exists, then the global minima of the energy function that is associated with the network represent such proofs. If no proof exist then the global minima indicate the lack of a proof. The network that is generated is of size polynomial in the bound k and the knowledge size. There are no restrictions on the type of logic formulas that can be represented. Most of the chapter discusses the representation of propositional formulas and proofs; however, an extension is presented that allows the representation of unrestricted first-order logic formulas (predicate calculus). Fault tolerance is obtained using a binding technique that dynamically assigns symbolic roles to winner takes all units.

A Fault Tolerant Connectionist Architecture for Construction of Logic Proofs

Gadi Pinkas

WUCS-93-10

March 1993

Department of Computer Science
Washington University
Campus Box 1045
One Brookings Drive
St. Louis MO 63130-4899

A Fault Tolerant Connectionist Architecture for Construction of Logic Proofs

Gadi Pinkas
November 15, 1992
WUCS-yy-n

Department of Computer Science
and Center for Optimization and Semantic Control
Washington University
509 Bryan, Campus Box 1045
One Brookings Drive
St. Louis, Missouri 63130
pinkas@cics.wustl.edu
Tel:(314) 567-4366

Abstract

This chapter considers the problems of expressing logic and constructing proofs in fault tolerant connectionist networks that are based on energy minimization. Given a first-order-logic knowledge base and a bound k, a symmetric network is constructed (like a Boltzman machine or a Hopfield network) that searches for a proof for a given query. If a resolution-based proof of length no longer than k exists, then the global minima of the energy function that is associated with the network represent such proofs. If no proof exist then the global minima indicate the lack of a proof. The network that is generated is of size polynomial in the bound k and the knowledge size. There are no restrictions on the type of logic formulas that can be represented. Most of the chapter discusses the representation of propositional formulas and proofs; however, an extension is presented that allows the representation of unrestricted first-order logic formulas (predicate calculus). Fault tolerance is obtained using a binding technique that dynamically assigns symbolic roles to winner takes all units.

Keywords: Logic, Fault Tolerance, Symmetric Network, Proofs, Constraint relaxation.

*Supported by NSF grant number: 22-1321-57136

1. Introduction

The ability to reason from acquired knowledge is undoubtedly one of the basic and most important components of human intelligence. Among the major tools for reasoning in the area of AI are deductive proof techniques. However, traditional methods are plagued by intractability, inability to learn and adjust, as well as by inability to cope with noise, faulty hardware and inconsistency. A connectionist approach may be the missing link: fine grain, massively parallel architecture may give us real-time approximation; networks are potentially trainable and adjustable; and they may be made tolerant to noise or unit faults as a result of their collective computation.

Most connectionist reasoning systems that implement parts of first-order logic (see for examples: [Hölldobler 90], [Shastri, Ajjanagadde 90]) use the spreading activation paradigm and usually trade[1] expressiveness with time efficiency. In contrast, this chapter uses the energy minimization paradigm (like [Derthick 88], [Ballard 86], [Pinkas 91] and [Anandan et al. 89]), representing an intractable problem, but trading time with correctness; i.e., as more time is given, the probability of converging to a correct answer (a global minimum) increases.

Symmetric connectionist networks used for constraint satisfaction are the target platform [Hopfield 82], [Hopfield 84], [Hinton, Sejnowski 86], [Peterson, Hartman 89], [Smolensky 86]. They are characterized by a symmetric matrix of weights and a quadratic energy function that should be minimized. Each unit of the network asynchronously computes the gradient of the energy function and changes its activation value so energy decreases and a stable state is found. Some of the models in the family of symmetric networks perform simulated annealing or deterministic annealing thus, they may be seen as performing a search for a *global* minimum of their energy function.

The task is therefore to represent logic deduction that is bound by a *finite* proof length[2] as energy minimization. The network should allow us to integrate in a unified way both knowledge about facts and rules, and meta-knowledge about the reasoning process itself. The inference mechanism (logic deduction) should be combined with the knowledge (predicate logic) into a single network of polynomial size such that when global minimum is found, a proof (if one exists) is also found. If no proof exists, the global minima represent the lack of a proof.

When a query is clamped, the network should search for a proof that supports the query. If a proof to the query exists, then every global minimum of the energy function associated with the network represents a proof. The network that is generated is of polynomial size in the knowledge size and in the proof length.

Special consideration is given in this chapter to the goal of providing fault tolerance. In this work I present a technique for composing complex structures that is inherently fault tolerant and which is not based merely on duplicating components and generating redundancy.

The chapter[3] is organized in the following way: Section 2 sketches the main idea while section 3 explains the binding g mechanism and how fault tolerance is obtained. Sections 4

[1]Trading complexity with expressive power is a well known tradeoff of symbolic systems [Levesque 84].
[2]Without a bound on the proof length, the problem is undecidable.
[3]The chapter is an based on [Pinkas 92].i

and 5 give the details of the representation and the constraints needed for the propositional case. Section 6 extends the technique to first-order predicate logic (FOL). Section 7 discusses soft constraints and section 8 concludes.

2. Main Idea

The idea is to construct a network whose visible units represent clauses and literals, and to use parallel constraint satisfaction to cause a proof for a posted query to emerge on the units. The proof is based on resolution steps [Robinson 65][4]; however, it is not by refutation and any deductive step may be used instead of resolution. A proof is a list of clauses ending with the query (goal) such that every clause used is either an original clause, a copy of a clause that appears earlier in the proof, or a result of a resolution step of the two clauses that appeared just earlier. At a global minimum, the activations of the visible units represent a list of clauses and unifications that together form a proof.

The network that is constructed is a set of units and weights that impose constraints upon these units.

The constraints encoded are of three classes: 1) Proof constraints: impose a resolution-based proof (resolution steps, copy steps and unifications); 2) Knowledge base constraints: impose the proof to use syntax of clauses from the knowledge base; 3) Soft constraints: cause "better" proofs to be preferred over not so "good" proofs ("better" will be defined later).

The first two groups are hard constraints that should be satisfied by exactly all valid proofs of length not higher than the given bound. The network performs a constraint optimization; it minimizes the violation of the soft constraint, while satisfying the hard constraints. When a state of the units is found that satisfies all the hard constraints and as many soft constraints as possible, then this state is a global minimum and it represents a valid and most desired proof. If no state satisfies all the hard constraints then no proof exists. If only the hard constraints are satisfied but the number of the satisfied soft constraints is not maximized, then we have a valid proof that may not be the "best" proof available; i.e., not the most general, not the shortest, etc,.

A matrix of units constitutes the proof area and functions as a clause list (see C in figure 2). This list represents an ordered set of clauses that form the proof. The query clauses are clamped onto this area and activate hard constraints that force the rest of the units of the matrix to form a valid proof (if it exists).

3. Structure Composition, Binding and Fault Tolerance

Creating complex data structures by binding objects together is a fundamental tool which is available in most programming languages and certainly in those used for symbolic computation. Of special importance for inference is the ability to perform dynamic variable binding

[4]Resolution is used for its simplicity and completeness. Other deduction steps may be used (like modus ponens) to gain efficiency.

(e.g. as in unification), where data structures need to be constructed and matched during run-time. Traditionally, dynamic generation of complex structures and variable bindings use pointers and concatenations. Unfortunately, such primitives are not available in our connectionist paradigm. Many connectionist variable bindings methods have been proposed recently (e.g., [Touretzky, Hinton 88], [Smolensky 87], [Lange, Dyer 89], [Shastri, Ajjanagadde 90])[5]; however, I found most of them not flexible enough for the dynamics and complexity of a logic reasoner.

The binding technique used is an extension of the method in [Anandan et al. 89] and may be viewed also as related to the variable binding technique used in [Barnden 91a]. In this technique, if two symbols need to be bound together, an instance is allocated from a pool of general purpose instances, and this instance is connected to both symbols. An instance can be connected to a literal in a clause, to a predicate type, to a constant, to a function or to a slot of another instance (for example, a constant may be bound to the first slot of a predicate). Variables are just slots of predicates or functions that are not bound by the user at query time.

In one simple form of the technique, a single matrix is used to bind two (or more) objects. We assume that no more than k different bindings will be needed at the same time in a proof,[6] and therefore, to handle bindings among n objects, we use a matrix of nk units. In Figure 1 the rows represent the objects and the columns represent instances. An object i allocates an instance j by setting the unit in row i and column j. The dark circles indicate units that are set. Two objects are bound if they share the same instance column; e.g., objects i and j are considered bound once they allocate the same instance l.

The rows of the matrix are winner-takes-all (WTA) networks that allow only one instance to be allocated per object.[7] By biasing each of the units in rows i and j we cause objects i and j to allocate an instance each. By specifying the hard constraint that rows i and j are equal, the two objects are forced to allocate the same instance and therefore to be bound. Note that any one of the k instances may be used for binding purposes. This property leads to robustness of the solution; if some of the units are faulty (stuck on either zero or one) and cannot satisfy the constraints, other units may be allocated that do satisfy the constraints. For example, if the unit at row i and column k is stuck on the value "zero", the system, in order to satisfy the constraints, may use the instance k' instead of k.[8]

We may use higher-dimensional matrices to represent attributes of the binding. For example, in a resolution-based proof, literals should be bound to clauses; however, each literal in a clause may be negated (negative) or unnegated (positive). Instead of maintaining two versions of every atomic proposition or predicate (as in [Ballard 86]), our solution is to assign a sign (negative/positive) as an attribute of the binding itself. Thus, positive and negative instances are used to bind literals to clauses. The sign attributes indicate whether the literal allocated via an instance is negated or unnegated. Matrix C of Figure 2 enables such bindings. The rows of the matrix represent clauses. Each instance that is active is allocated to a proposition (in matrix P), and the third dimension of C tells us whether this proposition is negated or not. The "+" sign inside the unit in row 3 and column 4 means that $C_{3,4,1}$ is active; i.e., a

[5]Almost every connectionist system for high-level reasoning must cope with the problem.
[6]In a proof whose length is bounded by k, the maximum number of bindings is also bounded by $O(k)$.
[7]Actually, the rows need not to be WTA; the WTA assumption is just a convenience.
[8]If the unit is stuck on "one", k is forced to be used as the binding instance.

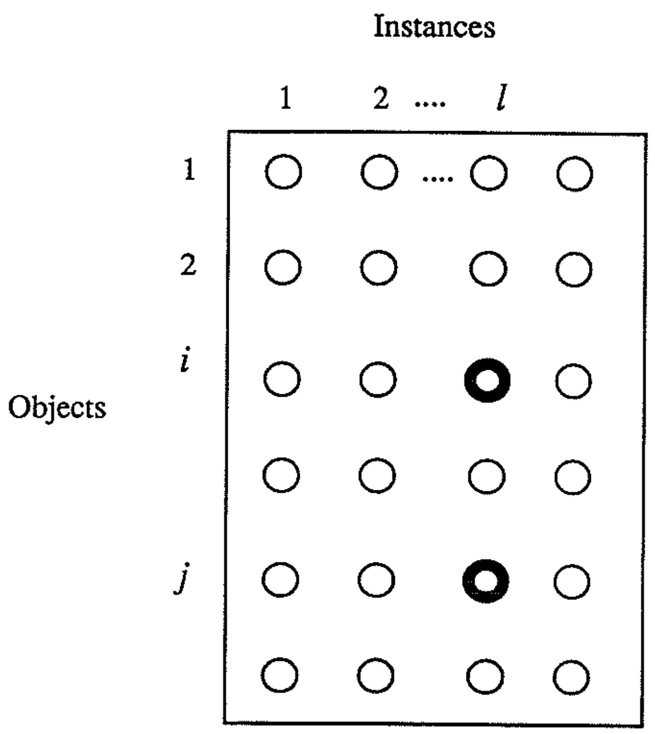

Figure 1: Simple binding: object i and object j are bound via instance l.

positive literal which is connected via instance 4, participates in clause 3. Similarly, the unit $C_{3,1,2}$ is active, which means that a negative literal also participates in clause 3.

Another example of the use of a three-dimensional matrix is for representation of nested structures. Here we bind together instances to slots of other instances. The objects are instances that are allocated to predicates and functions (in a different table). Each of the predicates/functions may have several slots where other functions may be bound. In matrix $NEST$ in Figure 4 the compound structure $R(f(b,a),b)$ is represented. The columns of $NEST$ represent instances allocated to predicates whose slots must be filled. The rows of $NEST$ represent instances that should fill these slots. The third dimension represents the slots that need to be filled. For example, the objects a, b, f, R allocate the instances 2,6,5,1 respectively in matrix P. The matrix $NEST$ is responsible for the binding of an instance to a slot of another instance. In the example, $NEST_{2,5,2}$ means that the constant a (instance 2) is bound to the second slot of the function f (instance 5). Instance 5, which represents $f(b,a)$, is bound to the first slot of instance 1 (predicate R) by activating unit $NEST_{5,1,1}$. Similarly, the constant b is bound to the second slot of R by activating $NEST_{6,1,2}$.

The mechanism that allows nesting is robust to hardware failures in the same sense as the simple case described earlier in this section. When a unit is stuck on zero, it cannot be used for binding purposes and another instance needs to be activated. Similarly, when a unit is stuck on one, it may still be used for binding. The system will try to satisfy the constraints that are imposed, with faulty units treated as additional constraints. Of course, some combinations of faulty units may contradict constraints and harm the system; however, the probability for such unique combinations of faults seems to be small. As a result, the number of instances available for binding will shrink gracefully as more random faults are introduced. Few random faulty units are unlikely to harm the system.

4. Representing proofs of propositional logic

I'll start by assuming that the knowledge base is propositional; i.e., each clause in the knowledge base is a list of positive and negative literals and each literal is an atomic proposition. Section 6 extends the mechanism to first-order predicate logic.

4.1. The proof area

The clauses that participate in the proof are represented using a 3-dimensional matrix ($C_{s,i,j}$) and a 2-dimensional matrix ($P_{i,j}$) as illustrated in figure 2. The rows of C represent clauses of the proof, while the rows of P represent atomic propositions. The columns of both matrices are the pool of instances used for binding propositions to clauses.

A clause is a list of negative and positive instances that represent literals. The instance thus behaves as a two-way pointer that binds composite structures like clauses with their constituents (the atomic propositions). If the same instance is allocated both to a clause and to an atomic proposition we say that the clause and the proposition are bound.

A row i in the matrix C, represents a clause which is composed of pairs of instances. If the unit $C_{+,i,j}$ is set, then the matrix represents a positive literal in clause i. If $P_{A,j}$ is also set, then $C_{+,i,j}$ represents a *positive* literal of clause i that is bound to the atomic proposition A. Similarly $C_{-,i,j}$ represents a *negative* literal.

A proof is a list of clauses that satisfies certain constraints. For example: given a knowledge base of the following clauses:
1) A
2) $\neg A \vee B \vee C$
3) $\neg B \vee D$
4) $\neg C \vee D$
we would like to prove the query D, by generating the following list of clauses (the order is the reverse of the common practice):

1) D	(obtained by resolution of clauses 2 and 3 by canceling A).
2) A	(original clause no. 1).
3) $\neg A \vee D$	(obtained by resolution of clauses 4 and 5 by canceling C).
4) $\neg C \vee D$	(original clause no. 4).
5) $\neg A \vee C \vee D$	(obtained by resolution of clauses 6 and 7 by canceling B).
6) $\neg B \vee D$	(original clause no. 3).
7) $\neg A \vee B \vee C$	(original clause no. 2).

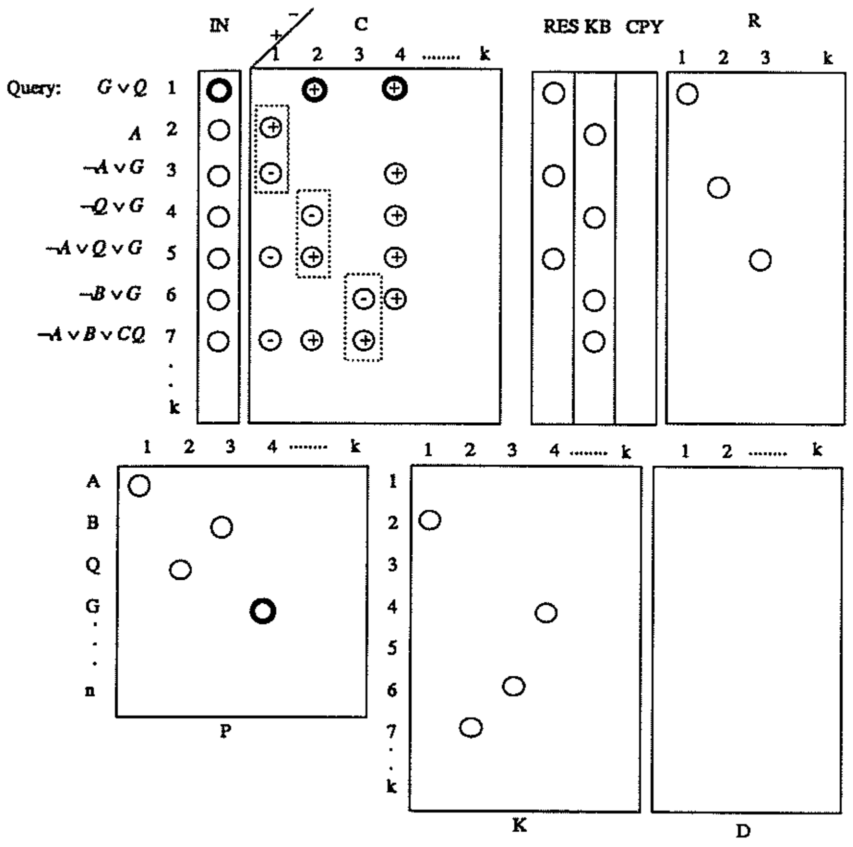

Figure 2: The proof area for a propositional case

Each clause in the proof is either an original clause, a copy of a clause from earlier in the proof, or a resolution step. The full representation of the above proof appears in figure 2.

In figure 2, the first row of matrix C is the query[9] clause D. It contains only one positive literal that is bound to atomic proposition D via instance 4. The third row of the matrix represents a clause of two literals: a positive one that is bound to D via instance 4, and a negative one bound to A via instance 1 (this is the clause $\neg A \vee D$); it is generated as a result of a resolution step.

4.2. Posting a query

A query is posted by clamping its clauses onto the first rows of C and setting the appropriate IN units. This indicates that the query clauses participate in the proof and should be proved by either a resolution step, a copy or an original clause.

Figure 2 represents the complete proof for D (the query). We start by allocating an instance (4) for D in the P matrix, and clamping a positive literal D in the first row of C ($C_{+,1,4}$); the

[9]If the query consists of more than one clause then it uses the first few rows of C.

rest of the row's units are clamped zero. The unit IN_1 is biased (to have the value of one) to indicate that the query is in the proof. If no proof exists, the IN_1 unit will become zero; i.e., the global minima is obtained by setting IN_1 to zero despite the bias.

Once a clause is in the proof, it must itself be proved and so on.

4.3. Participation in the proof

The vector IN represents whether a clause i participates in the proof. In our example, all the clauses are in the proof; however, in the general case some of the rows of C may be meaningless. When IN_i is on, it means that the clause i is in the proof and must be proved as well.

Every clause that participates in the proof is either a result of a resolution step (RES_i is set), a copy[10] of a some clause (CPY_i is set), or it is an original clause from the knowledge base (KB_i is set). Clause C_2 in figure 2 for example is an original clause of the knowledge base. If a clause j is copied it must be in the proof itself and therefore IN_j is set. Similarly, if it is a result of a resolution step (as in the case of figure 2) then the two clauses must also be in the proof ($IN_{i+1,j}$ and $IN_{i+2,j}$) and therefore must be themselves resolvents, copies or originals. This chain of constraints continues until all constraints are satisfied and a valid proof is generated.

4.4. Representing resolutions steps

The vector RES is a structure of units that indicates which are the clauses in C that are obtained by a resolution step; i.e., If RES_i is set, then the ith row is obtained by resolving row $i+1$ of C with row $i+2$. Thus, the unit RES_1 in figure 2 indicates that the clause D in the first row of C is a resolvent of the second and the third rows of C representing $\neg A \vee D$ and A respectfully. Two literals cancel each other if they have opposite signs and are represented by the same instance. In figure 2, the literal A in the third row of C and the literal $\neg A$ of the second row cancel each other generating the clause of the first row.

4.5. Representing canceled literals

The rows of matrix R represent literals canceled by resolution steps. If C_i is the result of a resolution step, there must be one and only one instance j such that both clause $i+1$ and clause $i+2$ include it with opposite signs. For example (figure 2): the clause D in the first row of C is the result of resolving the clause A with the clause $\neg A \vee D$ which are in the second and third rows of C respectfully. Instance 1, representing atomic proposition A, is the one that is canceled; $R_{1,1}$ is set therefore, indicating that clause 1 is the result of a resolution step that cancels the literals of instance 1.

[10]The proof example in figure 2 does not need copy steps; however, in general such copies are needed, if we want our proof mechanism to be complete.

4.6. Copied and original clauses

The matrix D indicates which clauses are copied to other clauses in the proof area. Setting $D_{i,j}$ means that clause i is obtained by copying clause j into clause i (figure 2 does not use any copy step).

The matrix K indicates which original knowledge-base clauses participate in the proof. The unit $K_{i,j}$ indicates that a clause i in the proof area is an original clause, and the syntax of the j-th clause in the knowledge base must be imposed on the units of clause i. In figure 2 for example, clause 2 in the proof assumes the identity of clause number 1 in the knowledge base and therefore $K_{1,2}$ is set.

5. Hard constraints

We are now ready to specify the hard constraints that must be satisfied by the units so that a proof is found. The constraints are specified as well formed logic formulas (boolean formulas). For example the formula $(A \vee B) \wedge C$ imposes a constraint over the units (A, B, C) such that the only possible valid assignments to those units are $(011), (101), (111)$. A general method to implement an arbitrary logical constraint on connectionist networks is shown in [Pinkas 90].[11]

5.1. In-proof constraints

If a clause participates in the proof, it must be either a result of a resolution step, a copy step or an original clause. In logic, the constraints may be expressed as:

$$\forall i : IN_i \rightarrow RES_i \vee CPY_i \vee KB_i$$

The three units (per clause i) consist a winner takes all subnetwork (WTA). This means that only one of the three units is actually set. The WTA constraints may be expressed as:
$RES_i \rightarrow \neg CPY_i \wedge \neg KB_i$
$CPY_i \rightarrow \neg RES_i \wedge \neg KB_i$
$KB_i \rightarrow \neg RES_i \wedge \neg CPY_i$

The WTA property may be enforced by inhibitory connections between every pair of the three units. Figure 3 illustrates an implementation of such subnetwork.[12]

5.2. Copy constraints

If CPY_i is set then clause i must be a copy of another clause j in the proof. This can be expressed as

$$\forall i : CPY_i \rightarrow \bigvee_j (D_{i,j} \wedge IN_j).$$

[11]The algorithm in [Pinkas 90] may generate hidden units if the constraint involve more than two units. The number of hidden units generated is in the order of the size of the formula.

[12]This is a general technique to implement a disjunctive constraint on WTA units. Many of the constraints in this chapter may be implemented in a similar way. Other techniques (not necessarily symmetric) may also be used.

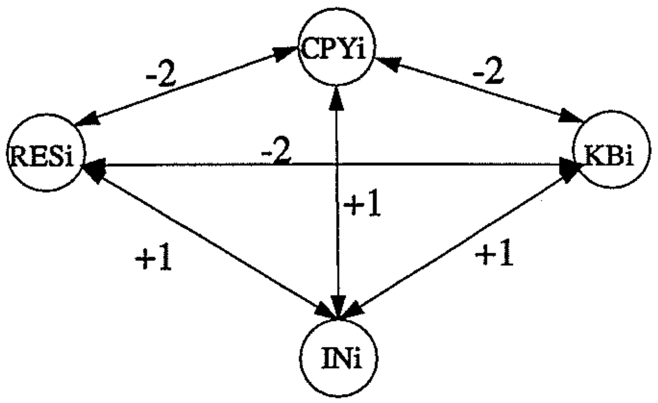

Figure 3: A disjunctive constraint on a WTA network ($IN_i \rightarrow (RES_i \vee CPY_i \vee KB_i)$).

The rows of D are WTA allowing i to be a copy of only one j; thus, the constraint may be implemented like the "in-proof" constraint mentioned earlier. In addition, if clause j is copied into clause i then every unit set in clause j must also be set in clause i.[13] This may be specified as:

$$\forall i,j,l : D_{i,j} \rightarrow ((C_{+,i,l} \leftarrow C_{+,j,l}) \wedge (C_{-,i,l} \leftarrow C_{-,j,l}))$$

5.3. Resolution constraints

If a clause i is a result of resolving the two clauses $i+1$ and $i+2$, then there must be one and only one instance (j) that is canceled (represented by $R_{i,j}$), and C_i is obtained by copying both the instances of C_{i+1} and C_{i+2}, without the instance j.

These constraints may be expressed as:

$\forall i : RES_i \rightarrow \bigvee_j R_{i,j}$	at least one instance is canceled
$\forall i,j,j',j' \neq j : R_{i,j} \rightarrow \neg R_{i,j'}$	only one instance is canceled (WTA)
$\forall i,j : R_{i,j} \rightarrow (C_{+,i+1,j} \wedge C_{-,i+2,j}) \vee (C_{-,i+1,j} \wedge C_{+,i+2,j})$	cancel literals with opposite signs.
$\forall i : RES_i \rightarrow IN_{i+1} \wedge IN_{i+2}$	the two resolvents are also in proof
$\forall i : RES_i \rightarrow (C_{+,i,j} \leftrightarrow (C_{+,i+1,j} \vee C_{+,i+2,j}) \wedge \neg R_{i,j}$	copy positive literals without the canceled
$\forall i : RES_i \rightarrow (C_{-,i,j} \leftrightarrow (C_{-,i+1,j} \vee C_{-,i+2,j}) \wedge \neg R_{i,j}$	copy negative literals without the canceled

5.4. Clause-instance constraints

The sign of an instance in a clause should be unique; therefore, any instance pair in the matrix C is WTA:

$$\forall i,j : C_{+,i,j} \rightarrow \neg C_{-,i,j}$$

[13]Clause i is either copied or weakened; i.e., more literals are added to the clause i in addition to the literals of j.

The columns of matrix P are WTA since an instance is allowed to represent only one atomic proposition:

$$\forall A, i, B \neq A : P_{A,i} \rightarrow \neg P_{B,i}.$$

The rows of P may be WTA:[14]

$$\forall A, i, j \neq i : P_{A,i} \rightarrow \neg P_{A,j}.$$

5.5. Knowledge base constraints

If a clause i is an original knowledge base clause, then there must be a clause j (out of the m original clauses) whose syntax is forced upon the units of the i-th row of matrix C. This constraint can be expressed as:

$$\forall i : KB_i \rightarrow \bigvee_{j}^{m} K_{i,j}$$

The rows of K are WTA networks so that only one original clause is forced on the units of clause i:

$$\forall i, j, j' \neq j : K_{i,j} \rightarrow \neg K_{i,j'}.$$

The only constraints that are left are those that force the syntax of a particular clause from the knowledge base. Assume for example that $K_{i,4}$ is set, meaning that clause i in C must have the syntax of the fourth clause in the knowledge base $(\neg C \vee D)$. Instances j and j' must be allocated to the atomic propositions C and D respectfully, and must appear also in clause i as the literals $C_{-,i,j}$ and $C_{+,i,j'}$.

The following constraints capture the syntax of $(\neg C \vee D)$:

$\forall i : K_{i,4} \rightarrow \bigvee_j (C_{-,i,j} \wedge P_{C,j})$	there exists a negative literal that is bound to C;
$\forall i : K_{i,4} \rightarrow \bigvee_j (D_{+,i,j} \wedge P_{C,j})$	there exists a positive literal bound to D.

Note that because the rows of P are WTA, it is possible to implement the constraints using :

$$\forall i, j : K_{i,4} \rightarrow (C_{-,i,j} \wedge P_{D,j})$$

$$\forall i, j : K_{i,4} \rightarrow (C_{+,i,j} \wedge P_{C,j}).$$

6. FOL extension

In first-order predicate logic instead of atomic propositions we must deal with predicates. As in the propositional case, a literal is represented by a positive or negative instance; however, an instance must be allocated to a predicate name and may have slots to be filled by other instances (representing functions and constants).

[14]This is not mandatory, but helpful in specify the syntactic constraints of the next subsection.

For example, an instance for the literal $R(f(X,a),X)$ must satisfy the following constraints: the instance (i) must be connected to a predicate type R; a second instance (j) that is connected to the function f must be connected also to the first slot of the instance of i. A third instance (k) that is connected to the constant a, must be also connected to the second slot of j. Finally, any instance that is connected to the first slot of j must also be connected to the second slot of i. To accommodate such complexity a new matrix $(NEST)$ is added, and the role of matrix P is revised (see figure 4).

The matrix P must accommodate now function names, predicate names and constant names instead of just atomic propositions. Each row of P represents a name, and the columns represent instances that are allocated to those names. The rows of P that are associated with predicates and functions may contain several different instances of the same predicate or function, thus, they are not WTA anymore.

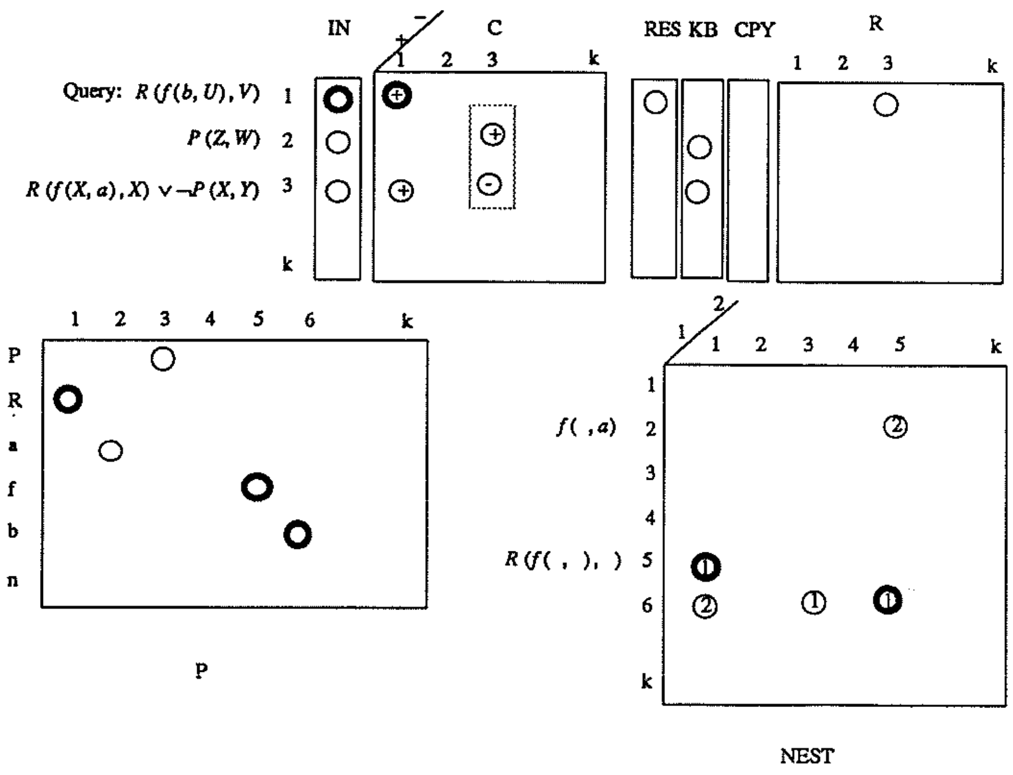

Figure 4: The proof area for a FOL example: proving $R(f(b,U),V)$ from $P(Z,W)$ and $R(f(X,a),X) \vee \neg P(X,Y)$.

In order to represent compound terms and predicates, instances may be bound to slots of other instances. The new matrix $(NEST_{i,j,p})$ is capable of representing such bindings. If $NEST_{i,j,p}$ is set, then instance i is bound to the p slot of instance j. Instance 1 in figure 4 for example, represents the predicate $R(f(X,a),X)$: instance 1 is allocated to predicate R;

instance 5 is allocated to the function f; instance 2 is allocated to the constant a and 6 is used as an instantiation of the variable X. Instance 5 (the function f) is bound to the first slot of instance 1 using the unit $NEST_{5,1,1}$; the constant a represented by instance 2 is bound to the second slot of 5 ($NEST_{2,5,2}$). The variable X represented by instance 6 is bound to the first slot of instance 5 and to the second slot of instance 1 ($NEST_{6,5,1}$ and $NEST_{6,1,2}$). Note that the columns of $NEST$ are WTA, allowing only one instance to be bound to a certain slot of another instance.

When a clause i is forced to have the syntax of some original clause l, syntactic constraints are triggered so that the literals of clause i become instantiated by the relevant predicates, functions, constants and variables imposed by clause l..

For example, assume that clause m in the knowledge base is of the form: $R(f(X,a),X) \vee \neg P(X,Y)$. If clause i in C must have the syntax of clause m of the knowledge base, there must be some positive literal (instance j) of predicate R; and a negative literal (instance l) of predicate P. The first slot of j must be instantiated by some instance j' representing f. Some instance j'' representing the constant a must instantiate the second slot of j'. In addition, any instance that fills either the first slot of j', the second slot of j or the first slot of l, must fill all of these slots (representing the variable X). This compound constraint can be expressed as:

$\forall : i : K_{i,m} \rightarrow \bigvee_j (C_{+,i,j} \wedge P_{R,j})$ there exists a positive literal R in i;
$\forall : i : K_{i,m} \rightarrow \bigvee_j (C_{-,i,j} \wedge P_{P,j})$ there exists a negative literal $\neg P$;
$\forall i,j : K_{i,m} \wedge C_{+,i,j} \wedge P_{R,j} \rightarrow \bigvee_{j'} (NEST_{j',j,1} \wedge P_{f,j'})$
there exists a function f in the first slot of R;
$\forall i,j,j' : K_{i,m} \wedge C_{+,i,j} \wedge NEST_{j',j,1} \wedge P_{R,j} \rightarrow \bigvee_{j''} (NEST_{j'',j',2} \wedge P_{a,j''})$
there exists a constant a in the second slot of f in R;
$\forall i,j,j',l' : K_{i,m} \wedge C_{+,i,j} \wedge P_{R,j} \wedge NEST_{j',j,1} \rightarrow (NEST_{l',j',1} \leftrightarrow NEST_{l',j,2})$
whatever instantiates the first slot of f in R also instantiates the second slot of R and vice versa ($R(f(X,),X)$);
$\forall i,j,l,l' : K_{i,m} \wedge C_{+,i,j} \wedge P_{R,i} \wedge C_{-,i,l} \wedge P_{P,l} \rightarrow (NEST_{l',j,2} \leftrightarrow NEST_{l',l,1})$
whatever instantiates the second slot of R also instantiates the first slot of P and vice versa ($R(,X) \vee P(X,)$).

The second slot of l represents an unbounded free variable Y. There is no constraint on this slot and it can be instantiated later by dynamic unification (or not be instantiated at all).

Unification is implicitly obtained if two predicates are instantiated using the same instance while still satisfying the constraints. When a resolution step is needed, the network tries to allocate the same instance to the two literals that need to cancel each other. If the syntactic constraints on the literals permit such sharing of an instance, then the attempt to share the instance is successful and a unification occurs.[15]

[15]Note that occur check is done implicitly since it is impossible to generate infinite nested trees when there is a bound on the number of instances.

6.1. An example

Figure 4 represents the proof of the query $R(f(b,U),V)$ (where U,V are existentially quantified), in one resolution step using the clauses: $P(Z,W)$ and $R(f(X,a),X) \vee \neg P(X,Y)$. The state of the proof area in figure 4 represents the proof with the necessary unifications (after a global minimum was reached):

1)$R(f(b,a),b)$	The query is obtained by resolving 2 and 3;
2)$P(b,W)$	an original clause;
3)$R(f(b,a),b) \vee \neg P(b,W)$	an original clause.

The query is clamped by allocating an instance (1) to R, and two other instances (5,6) to f and b respectfully ($P_{R,1}, P_{f,5}, P_{b,6}$). By clamping $NEST_{6,5,1}$ we specify that the constant b fills the first slot of f . By clamping $NEST_{5,1,1}$ we specify that the function f fills the first slot of R. The first clause of the proof area is clamped by setting $C_{+,1,1}$ while the rest of the units in the first clause are clamped to zero. To initiate the search, IN_1 is clamped.[16]

The system attempts to prove the query by allocating instance 1 (of the query) also to the literal $R(f(X,a),X)$. This causes instance 6 (the constant b) to fill also the second slot of 1 and the first slot of 3 which is the instance that happened to be allocated to the literal $\neg P(X,Y)$. This last instance (3) is shared with $P(Z,W)$, the two literals cancel each other and the query (represented by instance 1) is proved.

7. Optimization and soft constraints

The constraints discussed in the previous section are sufficient to force valid proofs on the units of the proof area; however, among the valid proofs some are preferable to others. By means of soft constraints and optimization it is possible to encourage the network to search for the preferred proofs. Theorem-proving thus is viewed as a constraint optimization problem. A weight may be assigned to each of the constraints[17] and the network tries to minimize the weighted sum of the violated constraints, so that the set of the optimized solutions is exactly the set of the preferred proofs. The following are some examples where this approach is useful.

7.1. Most general unifications

Two literals of different sign are unified iff they are instantiated by the same instance. The hard constraints imposed on the instances are enough to assure that the unification is correct if the constraints are satisfied. However, the unification that is obtained is not necessarily the most general one (MGU).

[16]Note that existentially quantified variables in the query are left with no constraints to limit them. Universally quantified query variables should be skolemized; i.e., instantiated by a unique constant that cannot be matched by any constant in the knowledge base. Remember that unlike standard resolution, the query is not negated and this is the reason we skolemize universal quantifiers and not existential quantifiers.

[17]For a general method for expressing weighted soft constraints on symmetric networks see [Pinkas 91].

The assignment of small penalties (negative bias) to every binding of a function (or a constant) to a position of another instance (in $NEST$), causes the network to search for a proof that uses as few bindings as possible:

$$\forall i, j, s : \epsilon(P_{f,j} \rightarrow \neg NEST_{i,j,s}).$$

The penalty (ϵ) is small enough not to interfere with the hard constraints; it is sufficient however, to cause the network to prefer unifications with fewer bindings. It is easy to show that a unification with as few variable bindings as possible is a MGU.

7.2. Parsimony, minimizing cost of plans and reliability

If shorter proofs should be preferred, soft constraints may be added that penalize every clause that participates in the proof:

$$\forall i : \epsilon(\neg IN_i).$$

Minimizing the penalty thus corresponds to finding the shortest proof.

If the knowledge is used to reason about plans (for example, in situation calculus), each of the actions is represented by a function and may be penalized using its cost. Let f be an action and let ρ_f be the cost of this action (eg., the time it takes to perform the action): the constraints

$$\forall f \in \text{ACTIONS}, i : \rho_f(\neg P_{f,i}).$$

cause the network to search for plans that have minimal sum of costs.

Each of the clauses in the knowledge base may have a reliability (or certainty) associated with it. We can therefore rank all possible proofs according to their reliability. Let ρ_i be a reliability measure that is associated with the ith clause; the constraint

$$\forall i, j : 1/\rho_j(\neg K_{i,j})$$

causes the network to search for proofs that are based on more reliable clauses.[18] Attaching penalties to beliefs is a useful techniques to represent nonmonotonic knowledge (see [Derthick 88], [Goldszmidt, et al. 90], [Pinkas 91]).

7.3. Coping with inconsistency

The last approach of having reliability measures (certainties) attached to beliefs may be extended for dealing with inconsistent knowledge bases. Thus, consistent subsets of the beliefs in the knowledge base may be used to prove contradicting conclusions. The ranking among proofs determined by the certainties may be used for preferring one possible conclusion rather than its negation. It is possible to build an inference mechanism that entails φ iff there exists a proof that supports φ and no better (or equally good) proof exists that contradicts φ.

[18]The reliability rank of a proof is the sum of $-1/\rho_i$ of all the clauses that participate in the proof. A proof is better iff its reliability is higher.

This notion of competition among contradicting proofs enables us to express defeasible rules, defaults, and exceptions, as well as to reason with noisy knowledge.

To construct this nonmonotonic mechanism we may use the same constraints that introduce certainties to beliefs are used. In addition, two subnetworks are constructed: one that searches for a proof for the query and the other that tries to prove its negation. Using the hard constraint $Q \leftrightarrow \neg Q$ the combined network is forced to either prove Q or its negation. By biasing $\neg Q$ a bit more than the biasing for Q, we cause the proof for Q to win if and only if it is strictly better than any proof for $\neg Q$.[19]

8. Summary

Given a finite set T of m clauses, where n is the number of different predicates, functions and constants, and given also a bound k over the proof length, we can generate a network that searches for a proof with length not longer[20] then k, for a clamped query Q. If a global minimum is found then an answer is given as to whether there exists such a proof and the proof (with MGU's) may be extracted from the state of the visible units. A nonmonotonic extension of the technique allows proving Q from an inconsistent knowledge base, if a proof for Q is found and no "better" proof exists that contradicts Q.

In the propositional case the network that is generated is of $O(k^2 + km + kn)$ units and $O(k^3 + km + kn)$ connections.[21] For predicate logic there are $O(k^3 + km + kn)$ units and connections, and we need to add $O(k^i m)$ connections and hidden units, where i is the maximal complexity-level[22] of the syntactic constraints. Note that if the the maximal proof length (k) is a small constant, the size of the network becomes linear in the size of the original knowledge base.

Most of the problems in an earlier approach [Ballard 86] are fixed: 1) We are not limited to unit resolution and the syntax of the FOL clauses is not restricted at all; 2) the network is compact and the representation of bindings (unifications) is efficient; 3) nesting of functions and multiple uses of rules are allowed; 4) only one relaxation phase is needed (and not two as in [Ballard 86]); 5) the network is capable of coping with inconsistency; 6) the query does not need to be negated and pre-wired; it can be clamped during query time; and 7) global minimum always corresponds to a proof if one exists (no loops).

As for performance, local minima still exist, so the task of finding a *global* minimum may be hard (note that we are trying to solve an inherently intractable problem). Hopes for an improved average-case performance are based on the massively parallel architecture, on training

[19]The approach can be implemented easily for the propositional case; however, in FOL an inconsistent set of clauses may be used to form a faulty argument. I have not found a way yet to integrate a consistency check on a FOL "proof" so that a consistent proof is found in a single relaxation phase.

[20]The bound k limits the number of clauses in the proof the number of literals that participate in the proof and the number of sub-terms that participate; i.e., the number of bindings.

[21]If only pairwise connections are allowed, more hidden units ($O(K^3 + km + kn)$) are needed.

[22]The complexity level of a syntactic constraint is determined by the number of different instances involved in it. It is a function of the nesting level of the terms within the clause and of the interdependencies among them.

algorithms that re-shape the energy surface, and on other advances in the dynamics of neural and constraint satisfaction networks.

The architecture discussed has a natural fault-tolerance capability: the units in matrices C, P, R and $NEST$, are dynamically allocated for binding purposes. When a unit becomes faulty, it simply cannot assume a role in the proof, and other units are allocated instead. Similarly, instances (like in matrices K, D) or clauses (like in C, IN, RES, KB) are simply ignored if they cannot be used, and other instances/clauses are used instead.

Acknowledgment: I wish to thank Dana Ballard, Rina Dechter, Peter Haddawy, Ron Loui and Dave Touretzky for helpful discussions. The work has been supported by NSF grant number: R-9008012.

References

[Ajjanagadde 89] V. Ajjanagadde, "Reasoning with function symbols in a connectionist system," *Proceedings of the Annual Conference of the Cognitive Science Society,* pp. 388-395, 1989.

[Anandan et al. 89] P. Anandan, S. Letovsky, E. Mjolsness, "Connectionist variable binding by optimization," *Proceedings of the 11th Cognitive Science Society* 1989.

[Ballard 86] D. H. Ballard "Parallel Logical Inference and Energy Minimization," *Proceedings of the 5th National Conference on Artificial Intelligence*, Philadelphia, pp. 203-208, 1986.

[Barnden 91a] J.A. Barnden, "Encoding complex symbolic data structures with some unusual connectionist techniques," in J.A Barnden and J.B. Pollack, *Advances in Connectionist and Neural Computation Theory 1,* High-level connectionist models, Ablex Publishing Corporation, 1991.

[Dechter, Pearl 88] R. Dechter, J. Pearl, "Network-based heuristics for constraint-satisfaction problems," *Artificial Intelligence 34*, pp. 1-38, 1988.

[Derthick 88] M. Derthick "Mundane reasoning by parallel constraint satisfaction," PhD thesis, CMU-CS-88-182 Carnegie Mellon University, Sept. 1988

[Goldszmidt, et al. 90] M. Goldszmidt, P. Morris, J. Pearl, "A maximum entropy approach to nonmonotonic reasoning," *Proceedings of AAAI*, pp 646-652, 1990.

[Hinton, Sejnowski 86] G.E Hinton and T.J. Sejnowski, "Learning and re-learning in Boltzman Machines," in J. L. McClelland and D. E. Rumelhart, *Parallel Distributed Processing: Explorations in The Microstructure of Cognition I,* pp. 282 - 317, MIT Press, 1986.

[Hölldobler 90] S. Hölldobler, "CHCL, a connectionist inference system for Horn logic based on connection method and using limited resources," International Computer Science Institute TR-90-042, 1990.

[Hopfield 82] J. J. Hopfield "Neural networks and physical systems with emergent collective computational abilities," *Proceedings of the National Academy of Sciences 79,* pp. 2554-2558, 1982.

[Hopfield 84] J. J. Hopfield "Neurons with graded response have collective computational properties like those of two-state neurons," *Proceedings of the National Academy of Sciences 81*, pp. 3088-3092, 1984.

[Lange, Dyer 89] T.E. Lange and M.G. Dyer, "High-level inferencing connectionist network," *Connection Science 1,* 2, pp. 181-217, 1989

[Levesque 84] H.J. Levesque, "A fundamental tradeoff in knowledge representation and reasoning," *Proceedings of CSCSI-84*, pp. 141-152, London, Ontario, 1984.

[Peterson, Hartman 89] C. Peterson, E. Hartman, "Explorations in mean-field theory learning algorithm," *Neural Networks 2,* 6, 1989.

[Pinkas 90] G. Pinkas, "Energy minimization and the satisfiability of propositional calculus," *Neural Computation 3,* 2, 1991. Also in Touretzky, D.S., Elman, J.L. Sejnowski, T.J. Hinton, G.E. (eds), *Proceedings of the 1990 Connectionist Models Summer School*, San Mateo, Morgan Kaufmann.

[Pinkas 91] G. Pinkas, "Propositional Non-Monotonic Reasoning and Inconsistency in Symmetric Neural Networks," *Proceedings of ithe 12th International Joint Conference on Artificial Intelligence*, Sydney, 1991.

[Pinkas 92] G. Pinkas, "Constructing proofs in symmetric networks," in J. E. Moody, S. J. Hanson, R. P. Lipmann (eds.), to appear in *Advances in Information Processing Systems 4* (NIPS), pp. 217-224, 1992.

[Robinson 65] J.A. Robinson, "A machine-oriented logic based on the resolution principle," *Journal of the Association for Computing Machinery 12*, 1, pp. 23-41, 1965.

[Shastri, Ajjanagadde 90] L. Shastri, V. Ajjanagadde, "From simple associations to systematic reasoning: A connectionist representation of rules, variables and dynamic bindings," technical report, University of Pennsylvania, Philadelphia, MS-CIS-90-05, 1990.

[Smolensky 86] P. Smolensky, "Information processing in dynamic systems: Foundations of harmony theory," in J.L.McClelland and D.E.Rumelhart, *Parallel Distributed Processing: Explorations in The Microstructure of Cognition I* , MIT Press, 1986.

[Smolensky 87] P. Smolensky, "A method for connectionist variable binding," TR: CU-CS-356-87, University of Colorado Boulder, 1987.

[Touretzky, Hinton 88] D.S Touretzky, G.E. Hinton "A distributed connectionist production system," *Cognitive Science 12,* 3, pp.423-466, 1988.