Parallel Logical Inference

Dana H. Ballard and Patrick J. Hayes
Computer Science Department and Cognitive Science Program
The University of Rochester
Rochester, NY 14627

February 1984

Abstract

The inference capabilities of humans suggest that they might be using algorithms with high degrees of parallelism. This paper develops a completely parallel connectionist inference mechanism. The mechanism handles obvious inferences, where each clause is only used once, but may be extendable to harder cases.

1. Motivation

The prospect of automating inferences has long been the goal of researchers in artificial intelligence. The most obvious advantage is a more compact representation of knowledge bases (KBs). Without inference ability all relevant facts must be explicitly represented in the KB. Using inference, only a subset of the facts need be explicitly represented, since the rest can be derived when required. However, despite the huge payoff, this goal has so far proved elusive. One reason for pessimism is that the known algorithms for reasoning fall into the class termed NP-complete. In a nutshell, this classification means that no better algorithms are known than ones that try out all the possibilities. For theorem proving, the number of possibilities can be open ended. In contrast to this pessimistic result stands human performance data. Psychologists have shown the following *performance result*: *a huge variety of forced-choice decisions can be made by human subjects in under a few hundred milliseconds.*

This is a huge discrepancy in results. The theoretical result implies that problems of even a modest size can overwhelm today's computers, whereas the practical tests show complex decision making in 100 - 400 ms. Furthermore, we know that humans bring huge numbers of facts to bear to solve a specific problem. Thus we are led to conclude that either: (1) humans do not make complex inferences; or (2) humans use a better algorithm and/or data structure. In this paper we explore the second possibility. Our aim is to show that theorem proving can be done using a parallel probabilistic relaxation algorithm. The algorithm requires that problems be formulated as the intersection of (possibly huge) numbers of local constraints represented in networks. The intersection process takes a worst-case time proportional to the diameter of the network but in practice often runs in constant time. Of course any machine of constant size will not be able to handle hard

theorems in constant time. However, our conjecture is that: *theorems that humans can solve in a few hundred millisconds have a constant time solution on a parallel machine.*

For many scientific applications, an inference mechanism that handles only the simpler cases, and fails in many cases, might not be useful. However, for human inference mechanisms, this may not be the case. The reason is that the human inference mechanism can be viewed as one component of several in a perception-action process. For example, in our model, if the inference mechanism fails to identify a visual object, one of the options available is to move closer and gather more data. Thus our goal is to develop an inference mechanism that allows many inferences to be made in parallel but may also fail in many cases.

A general formulation of theorem proving is that of Robinson [1965]: to prove S $=>$ W where S and W are sets of clauses, we attempt to show that S $\cup$ ~W is unsatisfiable. One classical way of doing this is to use *resolution.* Two clauses, P(x)Q(x) and ~P(a), can be resolved to produce Q(a). The process of constraining the bindings of variables in the clauses is known as *unification.* The resolution theorem proving technique resolves pairs of clauses with the objective of producing the null clause. If this is done, the unsatisfiability of the set of clauses S $\cup$ ~W has been demonstrated, and consequently the theorem S $=>$ W is true.

The approach has several important assumptions: (1) *clauses may be used only once*; (2) *the knowledge base must be logically consistent*; and (3) *the method uses a large network that must be preconnected.*

Our approach uses observations by Kowalski [1975] and Sickel [1976]. First we try and filter the clauses using various kinds of constraints. This filtering process is parallel and removes options that are not compatible with the constraints. Once this is done we resolve clauses in parallel. During the development, the reader must constantly keep in mind the nature of the result: it is not guaranteed to work, but the hope is that it will work in most cases.

The overall organization of our parallel inference is shown in Figure 1. The machine has three basic parts:

1) *Consistency Constraints.* The first part has the goal of activating a logically consistent set of constraints. This is the focus of other research, and we assume that the enterprise is successful.

2) *Inference Constraints.* Filtering constraints [Sickel, 1976; Kowalski, 1975] deactivate parts of the network that do not apply to the problem.

3) *Resolution.* The last part of the algorithm uses a second filtering technique based on resolution. In this phase, parts of the network are deactivated if they correspond to pairs of clauses that would resolve where one of the pair contains a single predicate. If the entire network can be deactivated in this way, a proof has been found; otherwise, the result is inconclusive.

The formulation of the algorithm is in terms of a connectionist network [Feldman and Ballard, 1982] using a recently-developed probabilistic relaxation algorithm [Hopfield, 1982]. One of the key contributions of this paper is to show that theorem proving can be described in terms of this formalism. The formalism has several advantages, but the main one is elegance: the problem can be described in terms of nodes which have binary states. During the course of the computation, constraints cause nodes to be turned off or on.

2. The Filtering Process

The objective of the filtering process is to define a set of local constraints that reflect the rules of predicate logic. Starting with a predicate logic formulation, we can examine the set of clauses and derive constraints that must hold between them, the predicate symbols, and the terms. These constraints are expressed in a common network formalism. The network consists of nodes which have binary states as described in the previous section. At each step in the filtering process the constraints for a particular node can be evaluated by evaluating that node's local input. If it cannot be part of the solution based on this local evaluation, it is turned off. The turning off of a node may cause other nodes to be turned off. This process converges when no more node state changes can be made.

The filter network has five sets of nodes: (1) C, the set of clause nodes; (2) P, the set of predicate letters and their complements; (3) F, the set of clause fragments; (4) B, the set of bindings between fragments; and (5) S, the set of substitutions. In any set of clauses there will be one cluase node, $c \in C$ for each clause in the set. There will be one clause fragment node $f \in F$ for each predicate letter mentioned in the clause. There will be a separate binder node $b \in B$ for each possible resolution between complementary predicates. Finally there will be a substitution node $s \in S$ for each possible substitution involving a binder. For example, in the following set S $= \{c_1$: P(x,a), c_2: $\neg$P(b,y)$\}$,

$$
\begin{aligned}
C &= \{c_1, c_2\} \\
P &= \{P, \neg P\} \\
F &= \{(c_1, P), (c_2, \neg P)\} \\
B &= \{((c_1, P)\ c_2, \neg P))\} \\
S &= \{xb, ya\}
\end{aligned}
$$

There are five different kinds of constraints: (1) a predicate letter constraint; (2) a clause-predicate substitution constraint; (3) a clause constraint; (4) unification constraints; and (5) a substitution constraint.

The Predicate Letter Constraint. The predicate letter constraint is derived from propositional logic. If in the set of clauses a predicate letter appears without its complement or vice versa, then that symbol can be pruned from the solution. In terms of the filter network, this constraint is easily expressed as an excitatory constraint between different nodes representing predicate letters, as shown in Figure 2. The weights and thresholds are arranged so that both the node and its complement must be on to keep each other turned on.

The Clause-Predicate-Substitution Constraint. This constraint is derived from the clauses in a straightforward way. Each clause may be decomposed into triples consisting of: (clause symbol, predicate letter, term). For example, C_1: P(x)Q(a) may be decomposed into (C_1, P, s_1) and (C_1, Q, s_2) where s_1 and s_2 are appropriate substitutions (these will be discussed further as part of the substitution constraints). In the filter network, there are a set of clause fragment nodes *F*, one for each triple. A clause fragment node *f* is connected to each node in the triple by mutually excitatory connections as shown in Figure 3. Its threshold is such that it will turn off if any of its constituents turns off.

The Clause Constraint. The clause constraint captures the notion that a clause can only be part of the solution if all of its fragments have viable bindings. Thus the fragments are connected to the node with a conjunctive connection. Figure 4 shows an example of a clause with three fragments. The conjunctive connection means that if any of the fragments are turned off, the clause will be turned off.

The Unification Constraints. The unification constraints capture possible bindings between terms. The clauses that can potentially resolve constrain possible bindings, and these possible bindings are realized by a set of binding nodes *B*. Bindings that are incompatible are connected by mutually inhibitory connections. Compatible bindings are connected by mutually excitatory connections. For example, in the set of clauses -P(a,b), P(x,y)Q(y,z), -Q(c,d), -P(a,c), the possible bindings are xa, yb, yc, and zd. Of these, compatible pairs are: (xa, yb), (xa, yc) and (yc, zd), and there is one incompatible pair: (yc, yd). This example is simple and does not capture all the constraints possible in unification. At least two others are necessary. These relate bindings between constants and variables. One is that if a variable is bound to a constant and another variable is bound to a different constant, then the two variables cannot be bound to each other. The other constraint is that if a variable is bound to a constant and the same variable is bound to a second variable, then the second variable can be bound to the constant. These constraints are summarized below:

$$x, y : \text{var} ; c, d \text{ const}$$

$$xc \,\&\, yd \Rightarrow \neg\, xy$$

$$xc \,\&\, xy \Rightarrow yc$$

$$xc \Rightarrow \neg\, xd$$

In the network there are potentially $|T|^2$ nodes where T is the set of literals used in the formulae to denote all possible variable-constant pairings. Thus the constraints in above are connected between all relevant groupings. A representative network fragment is shown in Figure 5.

Substitution Constraints. The possible substitutions constrain the network in two important ways. One additional constraint is necessary to link the different bindings together. In the logical formalism this constraint can be derived by observing the potential resolutions between clauses. (In Sickel's notation, these are arcs.) Substitution nodes *S* relate substitutions to bindings. The second constraint relates the substitution nodes to the clause fragment nodes. Each clause fragment node

mentions the terms in its predicate. If these terms are also mentioned in the substitution then there is a two-way positive link between the two nodes. Formally, a node s $\in S$ is positively linked to a node f $\in F$ if f is positively linked to a term $t_1 \in T$ and s is positively linked to a binding t_2x where t_2, x $\in$ T and $t_1 = t_2$. Figure 6a shows the assignment node to relate three bindings between two clause fragments. Since all the assignments must be satisfied, the connections *into* the assignment node are conjunctive.

3. The Resolution Process

The filtering constraints combine to reduce the network to a state where none of the bindings are inconsistent. If there are choices, they are decided arbitrarily. For example, the set $\{c_1$: P(x), c_2: ¬P(a), c_3: ¬P(b)$\}$ results in a network with two inconsistent substitutions: xa and xb. The probabilistic relaxation algorithm will make an arbitrary choice between these two possibilities. Thus the objective of the filtering process is to reduce the network to an *essential state*, wherein only one clause fragment can resolve with its complement. This might seem very restrictive, but the filter network can make many examples into this form. For example, the proof used by Henschen [1976] to introduce resolution can be reduced to this form without using resolution. However, the usefulness of this strategy will have to be tested with many different examples.

Once the network has been put into an essential state, the way constraints are handled can be changed slightly and the network will perform resolution. To do this, three changes are made:

1) the thresholds on clauses are changed so that singleton clause nodes are turned off;
2) the thresholds on clauses are changed so that dropping one fragment input does not turn off the node unless it is the last one; and
3) the binder network is fixed, so that no further changes take place.

The effect of turning off singleton clauses is to remove the fragment associated with their complements. Turning off all the nodes in this fashion is equivalent to finding a proof by resolution.

4. Examples

To describe the process, consider the example where S $\cup$ ¬W is given by: {P(x)Q(y)W(y), ¬W(z), ¬P(a), ¬Q(b)}. The network is shown in Figure 7. Note that all the clauses are essential. Removing any clause will cause all the nodes in the network to be turned off. For example, without clause C_4, f_6 is turned off. This could propagate as follows: ¬Q, Q, a_3, f_5, c_2, f_1, f_2, P, ¬P, s_1, s_2, c_1, f_4, W, ¬W. Of course, many other sequences are possible; the exact sequence in any given case depends on the probabilistic relaxation process.

Let us change the substitutions slightly and see what happens. Suppose c_2 is changed to P(z)Q(y)W(y). This modification is shown as a dotted line in Figure 7. Now the substitution constraint network comes into play. The network is prewired so that if za and yb are turned on, then they will turn off yz. Once this is done its effect will propagate through s_2 to turn off the entire network.

To continue the example, we now describe the resolution phase. Note that in this case turning of all the singleton clauses, c_2, c_3, and c_4, is sufficient to turn off the remaining clause c_1. Note that the clauses in {P(x)Q(y), W(y)¬P(a), ¬Q(b), ¬W(z)} can also be turned off in the same manner, but not those in {P(x)Q(y), W(y)¬P(a), ¬Q(b)¬W(z)}, which has no singletons, or those in {P(x)Q(y)W(y), ¬W(z)¬P(a), ¬Q(b)}.

5. Summary and Conclusions

The implementation of the first order logic constraints results in two coupled networks: (1) a clause network that represents the clause syntax; and (2) a binding network that represents the relationships between terms in different clauses. The method for resolving bindings, unification, can be as complex as the entire inference mechanism. Thus for our purposes we depend on the actual bindings in the KB to have a simple structure.

At the outset, the possibility of reusing clauses was ruled out, but there are some limited cases that can be handled. To see the necessity of reusing clauses, consider {SU ¬ W} = {C_1:P(a), C_2:P(b), C_3:¬P(x)Q(x), C_4:¬Q(a)¬Q(b)}. This can be handled by resolution in a straightforward way. The resolution tree is: ((C_1, C_3), ((C_2, C_3), C_4)). However, note that C_3 appears twice. The consequence of this is that since the unification constraints do not allow xa and xb simultaneously, the network will not pass the filter test. To handle this case we note that both possibilities for C_3 involve constant bindings. Thus we can resolve this by making two copies of C_3: ¬P(a)Q(a) and ¬P(b)Q(b). Once this is done, the inference mechanism will find the proof.

The main intent of this paper has been to force a new look at formal inference mechanisms from the standpoint of performance. Our contention is that models that do not have a parallel implementation are unlikely candidates for models of human inference. This realization may prove catalytic for approaches that try to unify the complementary goals of competence and performance.

The technical contribution of this paper is in the detailed specification of a network and inference mechanism. The network runs in parallel and can handle obvious inferences in first order logic. The running time is bounded from below by O(1) which occurs when all the constraints are local and O(diameter of network) which occurs when the constraints have to propagate the full extent of the network.

6. References

Feldman, J.A. and D.H. Ballard, "Connectionist models and their properties," *Cognitive Science* 6, 205-254, 1982.

Henschen, L.J., "A tutorial on resolution," *IEEE Trans. Computers C-25*, 8, 770-772, August 1976.

Hopfield, J.J., "Neural networks and physical systems with emergent collective computational abilities," *Proc., National Academy of Sciences USA 79*, 2554-2558, 1982.

Kowalski, R., "A proof procedure using connection graphs," *JACM 22*, 4, 572-595, 1975.

Robinson, J.A., "A machine-oriented logic based on the resolution principle," *JACM 12*, 1, 23-41, January 1965.

Sickel, S., "A search technique for clause interconectivity graphs," *IEEE Trans. Computers C-25*, 8, 823-835, August 1976.

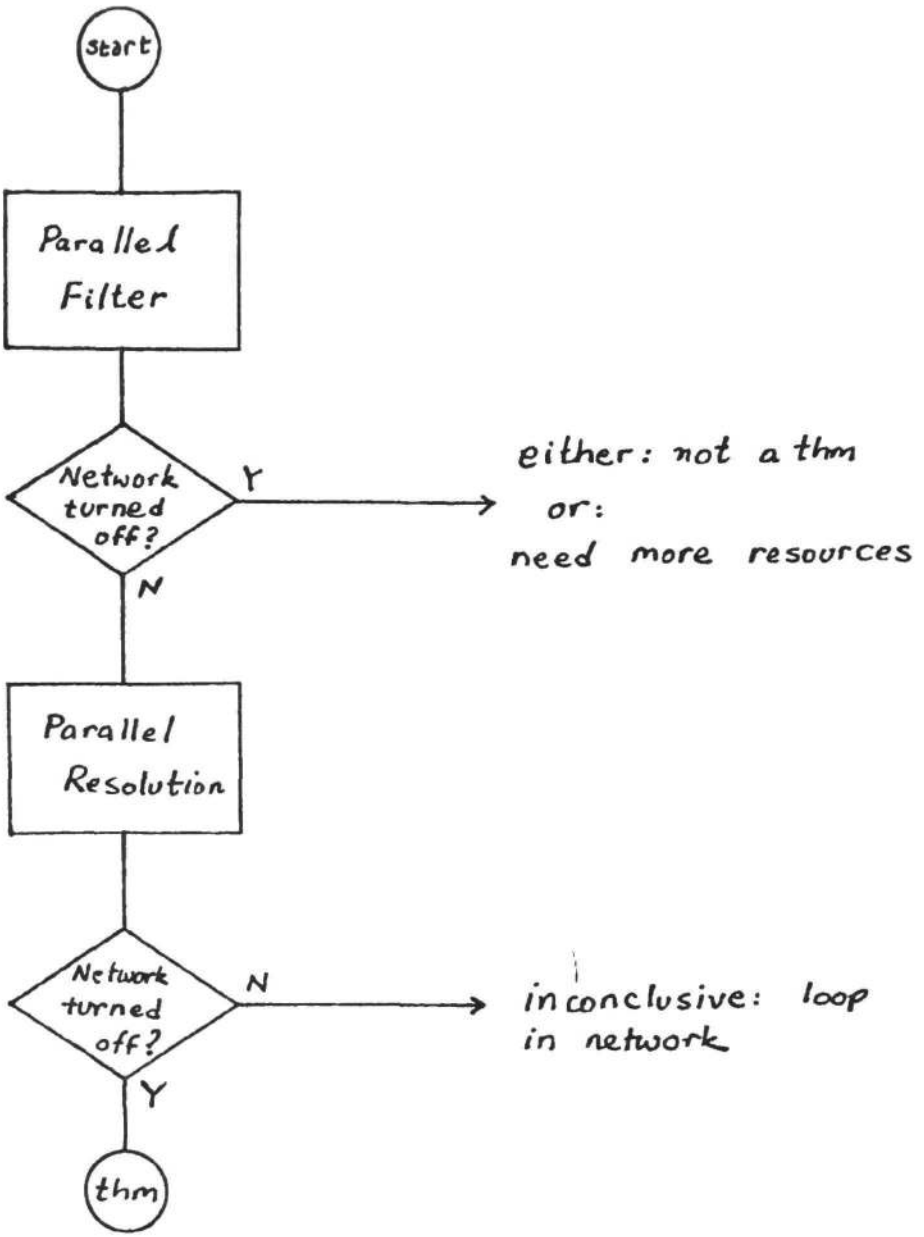

start
Parallel Filter
Network turned off?
Y
either: not a thm
or:
need more resources
N
Parallel Resolution
Network turned off?
N
inconclusive: loop in network
Y
thm

Fig. 1.

Fig. 2

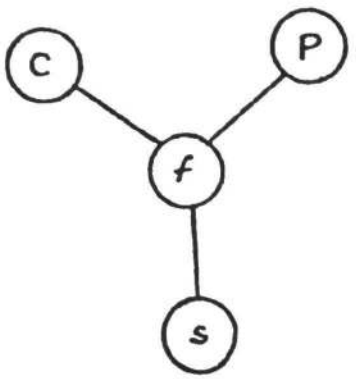


Fig. 3

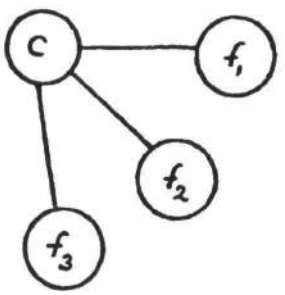
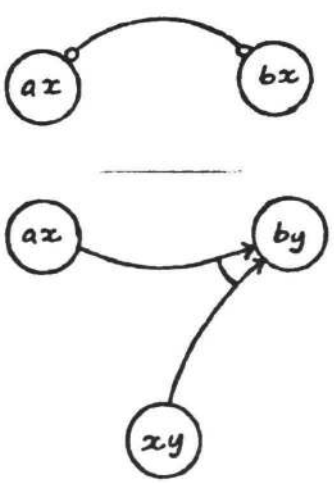


Fig. 4



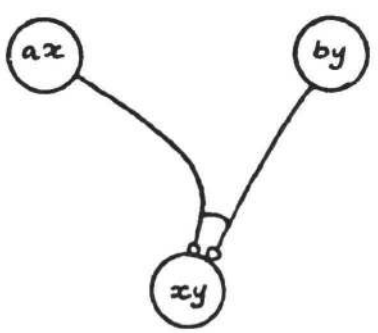
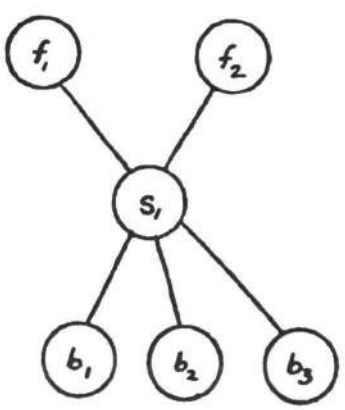


Fig. 5



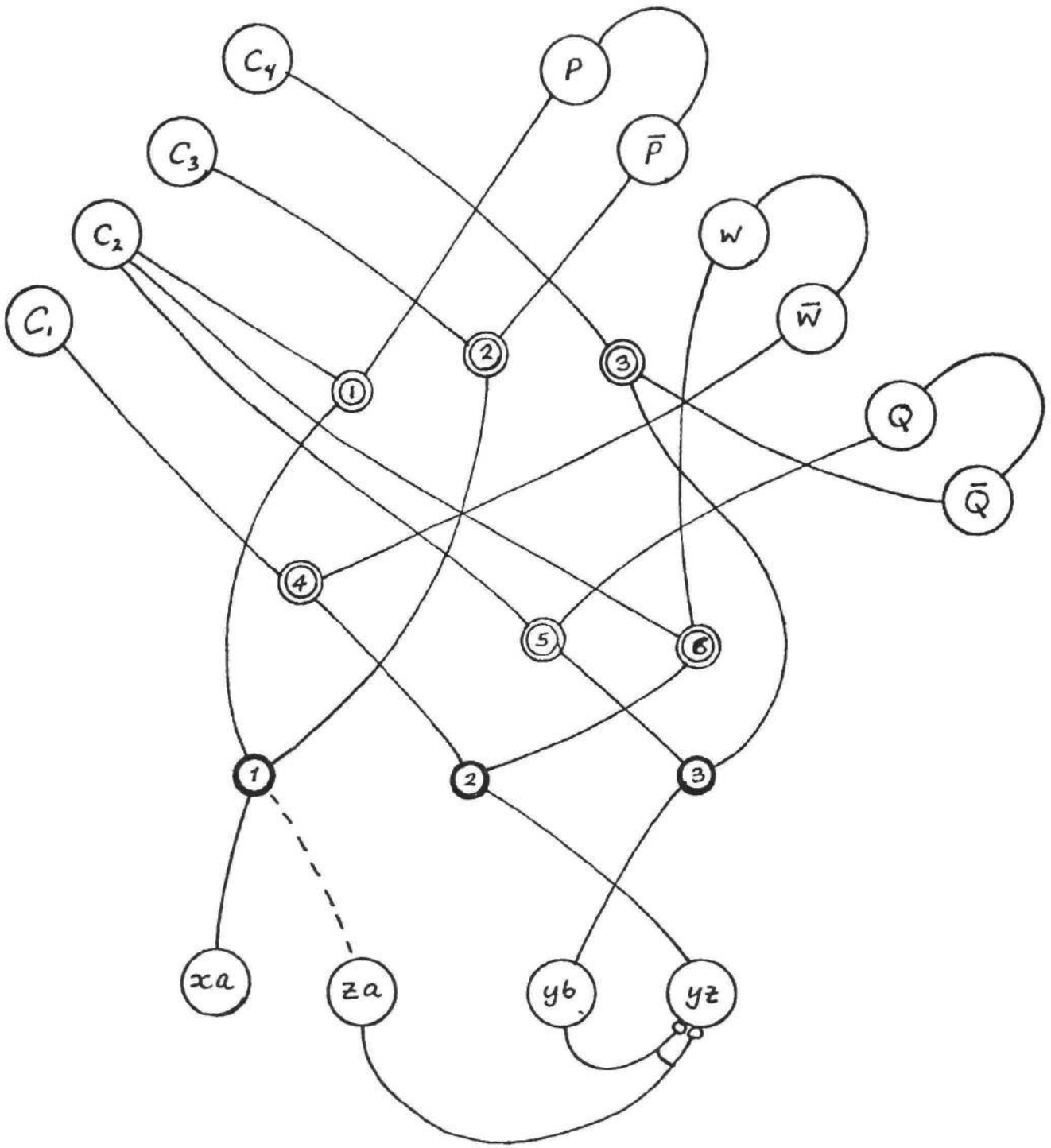


Fig. 7