CHCL - A Connectionist Inference System

Steffen Hölldobler
FG Intellektik, FB Informatik
TH Darmstadt
Alexanderstraße 10
D-6100 Darmstadt
Germany
xiisshoe@ddathd21.bitnet

Franz Kurfess
International Computer Science Institute
1947 Center Street
Suite 600
Berkeley, CA 94704
USA
kurfess@icsi.Berkeley.edu

TR-91-052
September 1991

Abstract

CHCL is a *c*onnectionist inference system for *H*orn logic which is based on the *c*onnection method and uses *l*imited resources. This paper gives an overview of the system and its implementation.

1 Introduction

No matter which definition of intelligent behavior one favors, the ability to draw conclusions from well-established facts certainly plays an important role. The formalization of this reasoning process has been the major goal in mathematical logic, leading to the development of various languages and calculi to express and formally compute the truth of statements. Automated inference systems based on predicate logic or derivatives and extensions thereof allow the computation of the truth of statements once these statements have been formulated. In this paper we present an inference system CHCL based on the connection method, a framework of calculi for the evaluation of predicate logic formulae. The execution of CHCL relies on connectionist techniques, giving rise to the exploitation of massive parallelism on one hand while opening the door towards the treatment of uncertain, incomplete and inconsistent information on the other hand.

The application of connectionist techniques to higher-level cognitive tasks such as reasoning has been suffering from a certain inadequacy in the representations used. The ease and elegance with which symbolic manipulation techniques are applied to dynamically changing data structures in conventional processing paradigms has not yet been matched with connectionist means (see [Güsgen and Hölldobler, 1991] in this book). One particular problem with respect to logic is the treatment of variable bindings; although some proposals to solve this problem have been made (see eg. [Ballard, 1986; Touretzky and Hinton, 1988; Shastri and Ajjanagadde, 1990b]), their integration into a logical framework is not fully satisfying. The approach proposed within CHCL relies on a full-fledged, distributed unification algorithm which computes the most general unifier for a set of terms [Hölldobler, 1990b]. The time required is linear to the size of the terms in the worst case, and constant (two steps) for important special cases like the word or the matching problem. The drawback of the algorithm is that it requires a number of nodes quadratic to the size of the terms in the worst case. The overall strategy pursued by CHCL is to identify so-called spanning matings in the formula under investigation, which represent potential alternative solutions. In order to really constitute a solution, the terms in these candidate matings have to be unifiable.

This article gives an overview of CHCL and its implementation. It is not concerned with technical details. They can be found in [Hölldobler, 1990a] and [Kurfeß, 1991b]. CHCL is based on Bibel's [1987] connection method and determines whether a set of Horn clauses is unsatisfiable. Therefore, we introduce the connection method for sets of Horn clauses in the following section 2. To avoid possible confusion between the connection method and a connectionist realization of the connection method, we will be very precise about the use of the word *connection*. A *connection* consists of a positive literal $\mathtt{P}(\mathtt{t}_1, \ldots, \mathtt{t}_n)$ and a negative literal $\neg\mathtt{P}(\mathtt{s}_1, \ldots, \mathtt{s}_n)$ with the same predicate symbol. In a figure literals defining a connection are connected by a curve. If in a connectionist network a unit directly excites or inhibits another unit, then there is a *link* between these units. Hence, *connection* always refers to the connection method and *link* refers to a connectionist network.

Section 3 gives an overview of the structure of CHCL and brief description of how a formula is represented and how queries are answered. A central part of each deductive system is the unification computation. In CHCL there is no restriction on the syntactic form of Horn clauses. Hence, the structured connectionist unification algorithm built into CHCL must be able to unify arbitrary first-order terms. This algorithm is presented in section 4. To prove the unsatisfiability of a given formula, we have to identify an appropriate set of connections (eg. [Stickel, 1987]). Bibel [1987] calls such sets of connections *spanning matings*. A spanning mating defines a proof iff all connected literals are simultaneously unifiable. In CHCL the spanning matings are computed one at a time and this process is described in section 5. As soon as a mating is selected all unification problems are solved in parallel. Section 6 presents the reduction techniques built into CHCL. These include the evaluation of isolated connections [Bibel, 1988] as well as the removal of non-unifiable or useless connections. In section 7 we modify CHCL such that binary constraint satisfaction problems can be solved efficiently. CHCL is currently been implemented at the International Computer Science Institute in Berkeley. Section 8 gives an account of this implementation. Finally, we conclude by outlining future work in section 9.

2 The Connection Method

The connection method is a formalism to compute the relations between different statements in a first-order language [Bibel, 1987]. It employs the propositional structure of a formula, which is defined by the various connections of the formula. A *connection* consists of a positive and a negative literal having the same predicate symbol. The central concept is a *spanning mating*, which is a set of connections such that the connections define a proof for the propositional structure of the formula. A proof for the first-order formula is obtained from a spanning mating if all connected atoms are simultaneously unifiable, in which case the mating is said to be *complementary*.

We have chosen the connection method since it allows for a global treatment of the whole search space, which – as we believe – is one of the main requirements for the development of massively parallel inference systems and intelligent global strategies. The method also separates the structure of a proof from the computation of the corresponding unifiers, which not only leads to a massively parallel computation of those unifiers but also allows the process of finding a proof to concentrate on the essential features of the given formula.

To illustrate the connection method (and its connectionist realization in the following sections) we consider a (simplified) program segment taken from the L_0 project at the International Computer Science Institute [Weber and Stolcke, 1990]. This project constitutes a recent effort in Cognitive Science to build a natural language aquisition system for a limited visual domain. The segment considered herein deals with spatial reasoning, namely the question of whether there is a circle X above a dark object Y.

```
        ?- object(obj(X,circle,_)), object(obj(Y,_,dark)), above(X,Y)

 II  I  object(obj(a,circle,dark)).   III
        object(obj(b,square,dark)).         IV
                                                          V
        above(Trajector,Landmark) :-
                   rAbove(Landmark,RegionAbove),
          VI       in(Trajector,RegionAbove).
```

```
rAbove(b,r1).
                    VII     VIII
in(a,r1).
in(a,r2).
```

In the visual scene shown in figure 1 we find a dark circle `a` and a dark square `b`. We know that a trajector is above a landmark if the trajector is in the region which is above the landmark. To keep the example small we simply assume that $\mathtt{r}_1$ is the region above the object `b` and that object `a` is in the striped region $\mathtt{r}_1$ as well as in the dotted region $\mathtt{r}_2$, whereas the simulation in [Weber and Stolcke, 1990] provides further rules for the predicates `rAbove` and `in` to describe these regions. The structure of the formula is given by the connections `I` to `VIII`. There are the eight spanning matings

```
{ I,     III,     V, VI, VII      },
{ I,     III,     V, VI,     VIII },
{ I,          IV, V, VI, VII      },
{ I,          IV, V, VI,     VIII },
{    II, III,     V, VI, VII      },
{    II, III,     V, VI,     VIII },
{    II,      IV, V, VI, VII      },
{    II,      IV, V, VI,     VIII }
```

defined by the various alternatives to solve subgoals which are engaged in more than one connection. But only the third mating $\{\mathtt{I},\mathtt{IV},\mathtt{V},\mathtt{VI},\mathtt{VII}\}$ is complementary and yields the bindings $\{\mathtt{X} \mapsto \mathtt{a},\ \mathtt{Y} \mapsto \mathtt{b}\}$ for the variables occurring in the initial query.

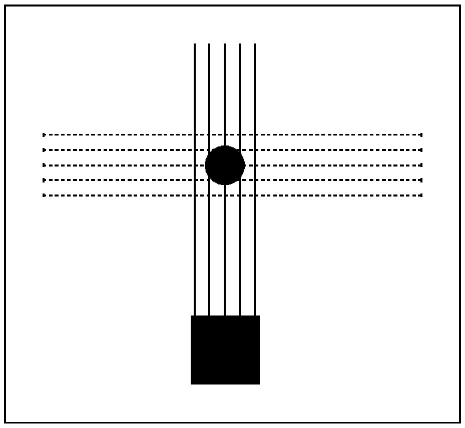

Figure 1: A dark circle `a` is above a dark square `b` and is in the striped and dotted regions.

3 An Overview of CHCL

CHCL uses Bibel's connection method. It generates systematically the spanning matings of a given formula one by one and unifies all connected atoms simultaneously. But CHCL is also a purely structured connectionist system in the sense of Feldman and Ballard [1982]. There is no central control or memory. The computing elements are simple neuron-like units which perform thresholding operations. They are interconnected through a network of weighted links, whose structure is determined by the given formula. The network is activated by clamping on certain inputs units. The activation is spread through the network until certain output units are activated indicating whether a proof of the formula has been found or not.

Figure 2 gives an overview of CHCL. The terms of the formula are represented in the term layer and the connections of the formula are represented in the connection layer. The computation starts by externally activating the `?`-unit. This will cause the spanning set layer to output the next spanning mating. Consequently, activation from the spanning set layer will spread through the connection layer and into the unification layer, where the connected atoms (viz. the corresponding terms) are simultaneously unified. More precisely, they are unified over the domain of rational trees [Colmerauer, 1982]. If the user wants to unify the terms over the domain of finite trees, then she or he has to activate the occur check request unit `OCR`, which will cause an occur check to be performed in the occur check layer. Depending on the outcome of the unification either the `Yes` or `No` unit will be activated. Repeated activations of the `?`-unit will cause the system to generate and test new spanning matings until an active `AG` unit indicates that all spanning sets were generated.

The above mentioned process describes the general procedure for the operation of the network. But before this general procedure is activated the formula will be reduced. By inspecting the possible spanning matings of our running example we find that each mating contains connections `V` and `VI`. In fact, these connections are *isolated* as the connected atoms are not engaged in any other connection ([Bibel, 1988]; see also subsection 6.3). In our example, the connections `V` and `VI` can be evaluated in parallel, the affected clauses can be replaced by their resolvents, and we obtain the following reduced formula.

```
        ?- object(obj(X,circle,_)), object(obj(b,_,dark)),
                                           in(X,obj(r1,region,striped)).
  II  I                                III
        object(obj(a,circle,dark)).        IV
        object(obj(b,square,dark)).                VII    VIII

        in(a,r1).
        in(a,r2).
```

But this is not the only reduction mechanism built into CHCL. Note that the atoms connected by `II` are not unifiable since the constants `circle` and `square` occupy the same position in these atoms. For similar reasons the atoms connected by `III` and `VIII` are also non-unifiable. All these non-unifiable atoms will simultaneously be detected in the reduction layer and, consequently, the connections `II`, `III`, and `VIII` will be eliminated. This leaves the connections `I`, `IV`, and `VII`. Since each of them is now isolated, they will be evaluated in parallel and the formula collapses to the empty clause. In other words, a proof for the formula was found by reduction techniques only and these techniques were applied in parallel.

In the subsequent sections we will describe in more detail how the various parts of the connectionist inference system are designed and how they work together.

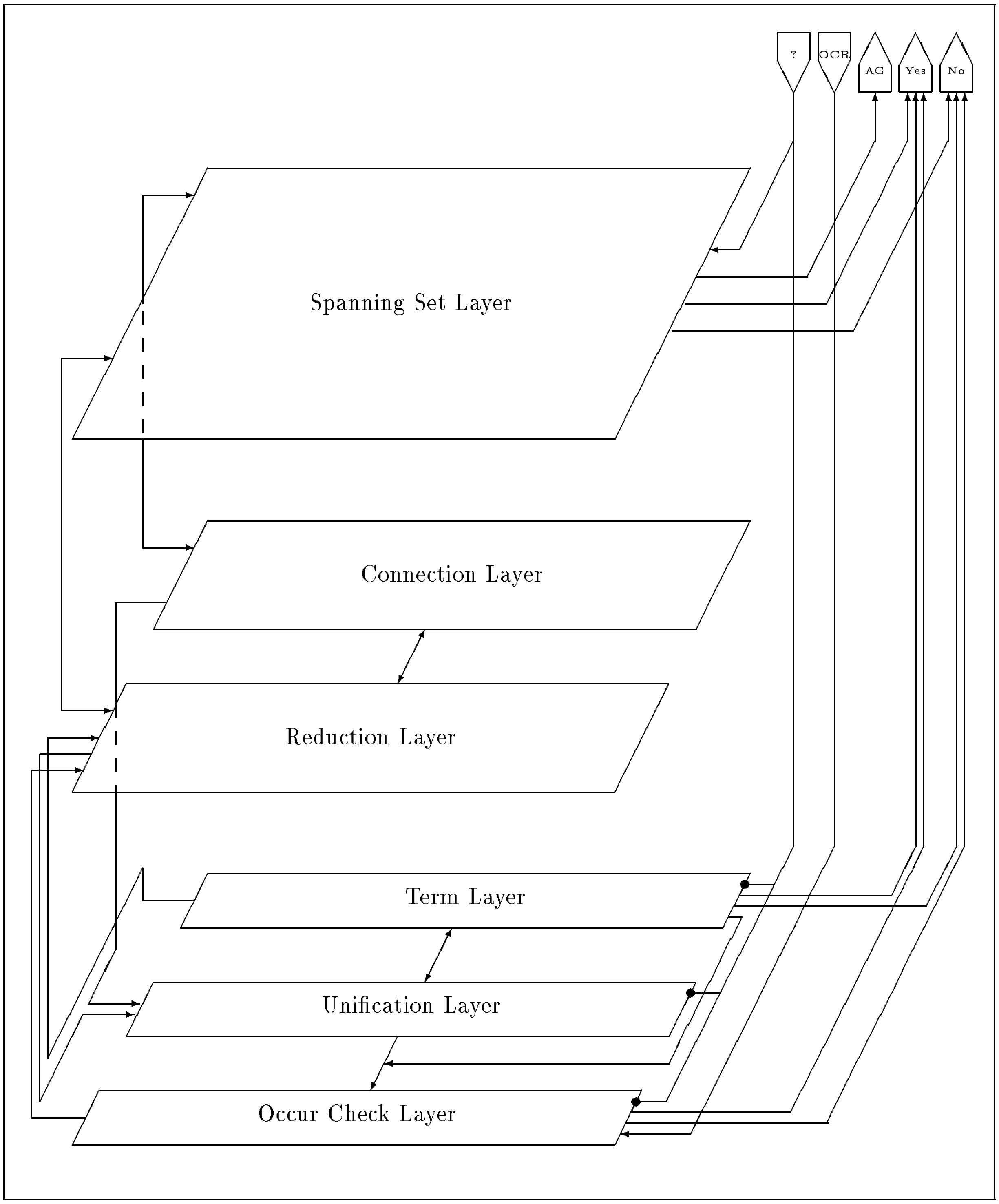

Figure 2: An overview of CHCL. An arrow from layer A to layer B indicates that units in A excite units in layer B. Similarily, a line with a bullet from A to B indicates that units in A inhibit units in B.

4 A Connectionist Unification Algorithm

At the heart of the inference system is a connectionist unification algorithm. The algorithm computes the most general unifier of two first-order terms as defined by Robinson [1965]. Terms and substitutions are represented by a set of position-label pairs. This representation is very similar to the role-filler representation used by Smolensky [1987]. For each position π and for each symbol $\mathtt{s}$ occurring in a term the term layer contains a unit $\langle\pi,\mathtt{s}\rangle$. A term like `obj(X,circle,dark)` can now be represented by externally activating the units $\langle\mathtt{0},\mathtt{obj}\rangle$, $\langle\mathtt{0.1},\mathtt{X}\rangle$, $\langle\mathtt{0.2},\mathtt{circle}\rangle$, and $\langle\mathtt{0.3},\mathtt{dark}\rangle$.

¿From [Paterson and Wegman, 1978] we know that the most general unifier of two terms can be obtained by computing a finest valid congruence relation on the representation of the terms. As shown in [Hölldobler, 1990b] this congruence relation is the closure of the operations

Singularity if $\langle\pi_1,\mathtt{X}\rangle$, $\langle\pi_2,\mathtt{X}\rangle$, and $\langle\pi_1,\mathtt{s}\rangle$ are active then activate $\langle\pi_2,\mathtt{s}\rangle$ and
Decomposition if $\langle\pi_1,\mathtt{X}\rangle$, $\langle\pi_2,\mathtt{X}\rangle$, and $\langle\pi_1.\pi,\mathtt{s}\rangle$ are active then activate $\langle\pi_2.\pi,\mathtt{s}\rangle$,

where π, π_1, and π_2 denote positions, $\mathtt{X}$ denotes a variable and $\mathtt{s}$ denotes a symbol. Informally, operation Singularity ensures that multiple occurrences of a variable are bound to the same term and operation Decomposition decomposes terms of the form $\mathtt{f}(\mathtt{t}_1,\ldots,\mathtt{t}_n)$ and $\mathtt{f}(\mathtt{s}_1,\ldots,\mathtt{s}_n)$ and forces the unification of the corresponding arguments $\mathtt{t}_i$ and $\mathtt{s}_i$, $1 \le \mathtt{i} \le \mathtt{n}$. It is quite straightforward to encode these operations in a structured connectionist network. This network is the unification layer in figure 2. Figure 3 shows as a simple example the representation of the terms $\mathtt{f}(\mathtt{X},\mathtt{X},\mathtt{X})$ and $\mathtt{f}(\mathtt{g}(\mathtt{a}),\mathtt{Y},\mathtt{g}(\mathtt{Z}))$ as well as the activation pattern obtained by unifying them. By checking the rows `0.1` and `0.1.1` we obtain the most general unifier $\{\mathtt{X}\mapsto\mathtt{g}(\mathtt{a}),\ \mathtt{Y}\mapsto\mathtt{g}(\mathtt{a}),\ \mathtt{Z}\mapsto\mathtt{a}\}$.

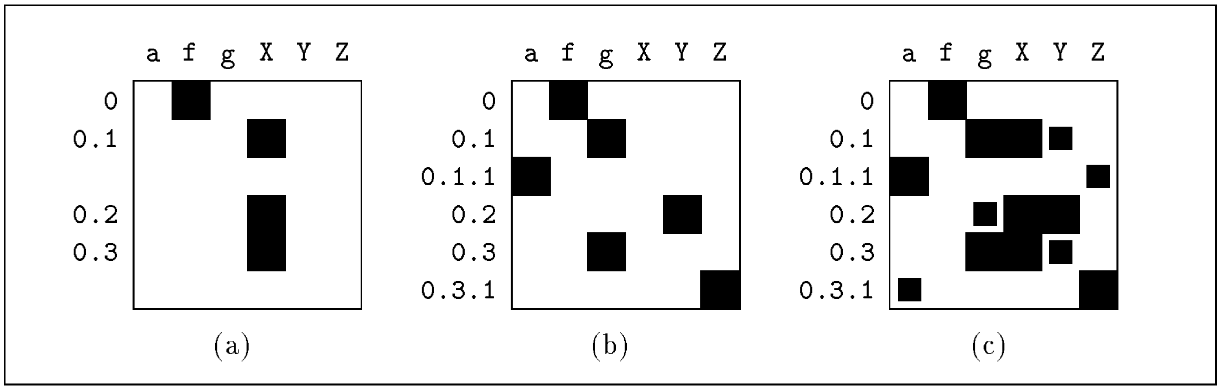

Figure 3: The representation of (a) $\mathtt{f}(\mathtt{X},\mathtt{X},\mathtt{X})$ and (b) $\mathtt{f}(\mathtt{g}(\mathtt{a}),\mathtt{Y},\mathtt{g}(\mathtt{Z}))$ in the term layer. Note that the matrix in (a) does not have a row for `0.1.1` and `0.3.1`. The two representations are merged in (c) and the closure of the operations Singularity and Decomposition is computed, which leads in one step to the activation of the units shown as small black boxes.

Not all unification problems admit a most general unifier. If during the unification computation two different function symbols occupy the same position, then the initial terms are not unifiable. The unification layer will detect such a case and the `NO` unit shown in figure 2 will be activated. The other reason where the unification computation may fail is in the presence of an occur check problem. This problem occurs if a variable `X` is to be bound to a term `t` which contains an occurrence of `X`. The unification algorithm within CHCL ignores this problem unless the user has activated the `OCR` unit shown in figure 2. In this case the occur check layer will eventually detect the problem and activate the `NO` unit.

[Hölldobler, 1990b] contains a formal definition of the term, unification, and occur check layer as well as a proof that the network computes the finest valid congruence relation for a unification problem if such a relation exists. The complete network for solving the unification problem requires $\mathtt{O}(\mathtt{N}^2)$ units and takes $\mathtt{O}(\mathtt{N})$ time in the worst case to settle down, where `N` is the number of positions in the unification problem. This behavior is not surprising since unification is known to be logspace-complete [Dwork *et al.*, 1984]. However, for important special cases such as the word or the matching problem, the network computes the solution in two steps and this behaviour is independent of the size of the unification problem.

The unification algorithm in CHCL does not unify one pair of terms at a time. Rather it simultaneously unifies all corresponding terms with respect to the connections in a selected spanning mating. Note that if all connections in a spanning mating define matching problems, then the unifiability of all connected atoms in the spanning mating is determined in two steps. This situation occurs – for example – whenever constraint satisfaction problems are solved within CHCL (see section 7).

5 The Selection of Spanning Matings

As described above the unification computation is triggered by the selected spanning mating. More precisely, for each connection in the given formula there is a unit called `spanning` in the connection layer. If a connection is in the currently selected spanning mating then its `spanning` unit will be activated. The `spanning` unit transmits the activation to the unification layer, where the connected atoms are unified.

But how is a spanning mating selected? The spanning layer shown in figure 2 can be regarded as a kind of structured Jordan [1986] or Elman [1989] network. It has an internal state `q` on which the two functions `next` and `output` operate. Whenever the user activates the `?`-unit, the function `output` generates the current spanning mating and `next` updates the state, ie.

$$\mathtt{spanning}_\mathtt{t} = \mathtt{output}(\mathtt{q}_\mathtt{t}, \mathtt{connected}_\mathtt{t}, ?)$$

and

$$\mathtt{q}_{\mathtt{t}+1} = \mathtt{next}(\mathtt{q}_\mathtt{t}, \mathtt{connected}_\mathtt{t}, ?),$$

where `spanning` denotes the generated spanning mating, `connected` denotes the active connections of the formula, and `t` denotes time. This process continues until all spanning matings are generated and tested, in which case `output` activates the `AG`-unit. The active connections are determined by the connection layer. Initially, each connection in the formula is active, which is indicated by an active `connected` unit in the connection layer. But some of the connections may not be a member of any spanning mating and, thus, should not be selected by the `output` function. One reason might be that the connected atoms are not unifiable. Reduction techniques built into CHCL and presented in section 6 will find those connections. They are thereafter removed by inhibiting their `connected` units in the connection layer.

Bibel's connection method generally requires to take copies of the clauses into consideration. Consequently, the set of spanning matings may be infinite. However, for various reasons discussed in section 9 the resources of CHCL are limited in that it can use only a fixed number of copies. In fact, for the purpose of this paper CHCL can handle only the original clause, where we assume that all clauses are standardized apart (ie. do not share common variables). Techniques for handling a fixed number of copies can be found in [Shastri and Ajjanagadde, 1990a]. Since there are now only finitely many different spanning matings, the logic is decidable.

The alert reader may have observed that finding a spanning mating now corresponds to deciding the satisfiability of the propositional structure of the given formula. Since all clauses are Horn, a sequential algorithm may find a spanning mating in time linear to the number `C` of connections in the formula [Dowling and Gallier, 1984][1]. In the spanning layer the connections of a formula are encoded as an AND-OR-tree, where the AND-branches correspond to the various conditions of a rule and the OR-branches correspond to the various alternatives to solve a condition. An activated `?`-unit excites the root of the tree, from where the activation spreads through the tree such that

- at each encountered OR-node one and only one alternative is selected and
- at each encountered AND-node all branches are selected.

The encountered leaves of the tree constitute the selected spanning mating (viz. the result of the function `output`). The function `next` decides which OR-branch is selected such that each time a different spanning mating is selected. Consequently, in the worst case it takes time linear to the depth of the AND-OR-tree to select a spanning mating. If the tree is balanced a spanning mating will be generated in time $\mathtt{O}(log(\mathtt{C}))$. On the other hand, the depth of the AND-OR-tree may be linear to `C`, in which case the tree is a chain. One might expect that this defines a worst case situation for the computation of the spanning mating. However, if the AND-OR-tree degenerates to a chain, then all connections are isolated [Bibel, 1988] and the respective reduction techniques presented in the following subsection 6.3 will generate the corresponding spanning mating in one step. For more details the reader is referred to [Hölldobler, 1990a], where all units together with their thresholds and weighted links in the spanning layer are defined.

6 Reduction Techniques

As the name indicates reduction techniques are used to reduce the search space. They can typically be applied in parallel and in time linear to the size (viz. the number of connections) of the formula. Several of these techniques are built into CHCL and they are presented in this section. However, we can give only an informal account of the various techniques and their realization in CHCL. All details can be found in [Hölldobler, 1990a].

[1]See also [Scutella, 1990] for a correction of a bug in Dowling's and Gallier's algorithm.

6.1 Removal of Non-unifiable Connections

A connection cannot participate in any proof if the connected atoms are not unifiable. Hence, the obvious technique would be to test all connections in advance. Unfortunately, since the representation of each term is such that for each position in the term there is precisely one row in the term layer and since unifiers are computed on this representation, we cannot perform this test in parallel. It would be a test of whether all connected atoms in the formula are simultaneously unifiable. Usually, this would lead to cross-talk if a variable occurs in an atom which is engaged in more than one connection and is bound to different terms in the various connections. There are cases where such a cross-talk cannot occur and these are treated in subsection 6.3.

However, it is easy to detect cases where atoms are not unifiable because different function symbols occur at the same position in these atoms. As an example consider the atoms

`object(obj(X, circle, _))`

and

`object(obj(b, square, dark))`,

which are connected by connection `II` in our running example. Since the constants `circle` and `square` occur both at position `0.1.2`, these atoms can never be unified. Chcl determines this non-unifiability with the help of the reduction layer and, consequently, the `connected` unit for `II` in the connection layer is inhibited. As described in section 5 this essentially removes connection `II` from the formula. This simplified test for the unifiability of connected atoms takes four steps and is performed for all connected atoms in parallel.

6.2 Removal of Useless Connections

The technique to remove useless connections is based on the close correspondence between propositional Horn formulas and context-free grammars (cf. [Dowling and Gallier, 1984]). In context-free grammars a non-terminal symbol is said to be useless if it does not occur in a sentenial form or cannot generate a terminal string (cf. [Aho and Ullman, 1972]). Correspondingly, we will call the head of a clause (viz. each connection of the head) *useless* if it does not occur in an SLD-resolvent of the initial clause or if the conditions of the clause cannot be solved. The important property of useless non-terminal symbols and connections is that they can be determined statically by analyzing the set of productions and clauses, respectively. Algorithms for context-free grammars can be found in [Aho and Ullman, 1972].

Informally, these algorithms are encoded in the following rules, where we assume that the initial formula does not contain any useless connections.

- If the head of a clause is not engaged in any connection (ie. if all `connected` units for connections with the head are deactivated), then inhibit all `connected` units for the connections with conditions of the clause.
- If a condition in a clause is not engaged in any connection (ie. if all `connected` units for connections with the condition are deactivated), then inhibit all `connected` units for the connections with the head of the clause.

These rules can easily be realized in a structured connectionist setting.

6.3 Evaluation of Isolated Connections

As mentioned in subsection 6.1 we can generally not test in parallel whether connected atoms are unifiable. However, besides the simple test in subsection 6.1 there is another exception which is due to [Bibel, 1988]. A connection is said to be *isolated*

- if the connected atoms are not engaged in any other connection, or
- if the connected atoms are ground (ie. do not contain variables), or
- if one of the connected atoms is ground and the other atom is not engaged in any other connection.

The rationale behind this definition is that atoms engaged in isolated connections can be unified in parallel without causing any cross-talk or backtracking. There is no alternative to the bindings generated by unifying isolated connections.

¿From our chosen representation of terms in the term layer and connections in the connection layer it is straightforward to encode these rules in a structured connectionist network such that all isolated connections are evaluated simultaneously. If a connection is found to be isolated, then the respective parts of the unification layer are activated forcing the unification of corresponding terms. If the unification fails, then the connection will be useless and its `connected` unit in the connection layer will be inhibited.

Our running example has initially the isolated connections `V` and `VII`. As shown in section 3 both connections (viz. connected atoms) unify simultaneously and the formula was reduced by replacing the clauses by their resolvents. The latter process will be described in more detail in the following subsection.

6.4 Removal of Solved Connections

Under certain conditions isolated connections can be removed by replacing the connected clauses by their resolvents. This is the case if the connections are solved. The motivation for this technique stems from the observation that a condition `A` in a clause can be eliminated if there is a unique minimal substitution σ such that σ`A` is a logical consequence of the given program and if the minimal substitution is applied to the remainder of the clause.

More formally, a connection between subgoal `A'` and head `A` of clause `C` is said to be *solved* if

- the connection is isolated,
- the atoms `A'` and `A` are unifiable, and
- each subgoal occurring in `C` is solved.

Obviously, a solved connection may participate in any spanning mating. As before, these rules can easily be encoded in a structured connectionist system.

In our running example we find that initially only connection `VI` is solved, whereas connection `V` is isolated and unifiable, but the `above`-clause contains unsolved connections engaged with its conditions. Fortunately, as soon as connection `VI` is unified, the variable `RegionAbove` is bound to the region `obj(r1,region,striped)`. Consequently, connection `VIII` is found to be non-unifiable by the technique described in subsection 6.1 and will be eliminated. Thus, connection `VII` becomes isolated as well as solved and, now, connection `V` is also solved. At the same time a similar process leads to the discovery that connections `I` and `IV` are also solved. In other words, the complete formula was solved by reduction techniques only and it was found that the circle `a` is above the square `b` in the scene of figure 1.

7 Constraint Satisfaction in CHCL

As finite constraint satisfaction problems can be stated as satisfiabilty problems in propositional Horn logic[2], CHCL can easily handle them. However, CHCL is not especially adapted for constraint satisfaction problems and will not handle them efficiently. In particular, CHCL does not perform a check for arc consistency. In [Hölldobler and Hower, 1991] it was shown how a slight modification of CHCL achieves arc consistency. This section contains a brief account of this modification.

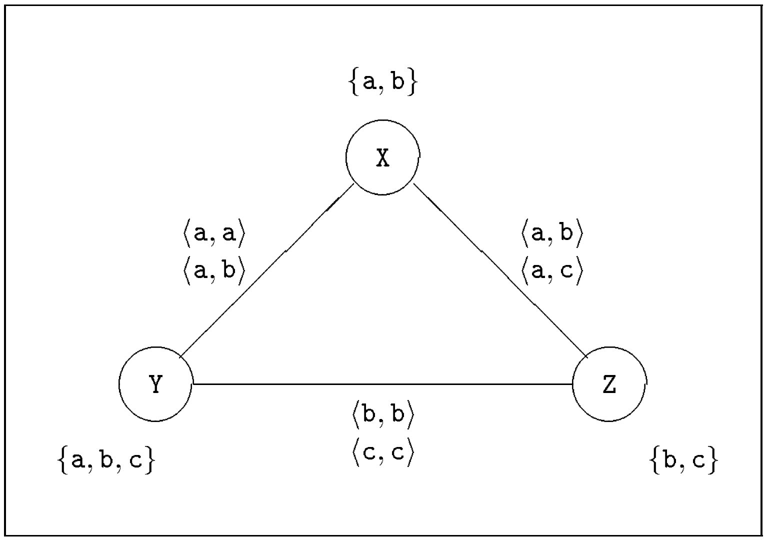

Figure 4: The constraint graph for a simple constraint satisfaction problem. The sets denote the domains (viz. unary constraints) of the variables and the tuples denote the binary constraints between corresponding variables.

As an example consider the simple constraint satisfaction problem shown in figure 4 as a constraint graph. The goal is to find bindings for the variables `X`, `Y`, and `Z` such that all unary and binary constraints are simultaneously fulfilled. This problem can be expressed by the following formula, where the `D`-clauses express the domain facts and the `C`-clauses express the binary constraints.

[2]See [Mackworth, 1987] for an introduction to constraint satisfaction.

```
?- CXY(X,Y), CXZ(X,Z), CYZ(Y,Z)

CXY(a,a) :- DX(a), DY(a)
CXY(a,b) :- DX(a), DY(b)

CXZ(a,b) :- DX(a), DZ(b)
CXZ(a,c) :- DX(a), DZ(c)

CYZ(b,b) :- DY(b), DZ(b)
CYZ(c,c) :- DY(c), DZ(c)

DX(a)  DX(b)   DY(a)  DY(b)  DY(c)   DZ(b)  DZ(c)
```

All connections between domain facts and the conditions of the constraint rules are isolated and will be evaluated simultaneously using the technique described in subsection 6.3. In fact, the connections are either non-unifiable or solved and the formula reduces to the following set of clauses.

```
?- CXY(X,Y), CXZ(X,Z), CYZ(Y,Z)

CXY(a,a)
CXY(a,b)

CXZ(a,b)
CXZ(a,c)

CYZ(b,b)
CYZ(c,c)
```

Since none of the reduction techniques described in section 6 is now applicable anymore, CHCL – as specified so far – has to generate the eight different spanning sets and to test whether the connected atoms are simultaneously unifiable. Though all unification problems encountered in a constraint satisfaction problems are matching problems and, hence, the unification is performed in two steps, this behavior is far from being optimal since the example can be solved by local constraint propagation techniques without any search.

The technique needed for solving our example is the test for arc consistency. *Arc consistency* states that a variable `X` may have value `a` if for each other variable `Y` there is a value `b` such that $\langle$`a,b`$\rangle$ is a valid binary constraint between `X` and `Y` (viz. `CXY(a,b)` is true). The arc consistency condition allows to remove any value from the domain of `X` which is unsupported.

In our example, we find that the value `b` in the domain of `X` as well as the value `a` in the domain of `Y` is unsupported. Hence, these values should be removed. In analogy to subsection 6.2 this can be done in CHCL by inhibiting all `connected` units for connections engaged with `DX(b)` and `DY(a)`. As a consequence, the formula may now contain useless connections. The interested reader is encouraged to apply the various reduction techniques presented in section 6 to the example. He or she will find that the formula eventually collapses to the empty clause generating the substitution $\{\mathtt{X} \mapsto \mathtt{a},\ \mathtt{Y} \mapsto \mathtt{b},\ \mathtt{Z} \mapsto \mathtt{b}\}$.

To implement the arc consistency condition we need an additional OR-of-AND unit in the sense of [Feldman and Ballard, 1982] for each domain fact. Such a unit will be activated if there is a variable which does not support the value anymore. This solution is very similar to the encoding described by Cooper and Swain [1988] and used by Güsgen [1990]. But whereas Cooper's and Swain's system is restricted to local satisfaction problems, our system can handle global constraint satisfaction problems as well. Güsgen's system may also handle global constraint satisfaction problems. But his units transmit Gödel numbers and compute greatest lower bounds and least common multiples. These messages and operations are fairly complex and usually not allowed in truly connectionist systems (cf. [Feldman and Ballard, 1982]). An alternative encoding of the Gödel numbers as bit vectors, however, increases the number of units in the system considerably.

8 Implementation

8.1 ICSIM

The implementation of CHCL currently pursued at the International Computer Science Institute is based on a connectionist simulator called ICSIM, developed at the same institution by H.-W. Schmidt [1990]. ICSIM provides a set of basic building blocks for connectionist networks as a collection of library classes. It is implemented in EIFFEL [Meyer, 1988], an object-oriented language and development environment. The use of an objet-oriented methodology allows for both flexibility and reuse of basic classes and enables the user to get an easy start by mainly relying on the classes provided through the library as well as modifying some of the classes and their routines for efficient customized implementations. ICSIM currently is being ported to SATHER, an offspring of EIFFEL aimed at simplification and more efficient execution [Omohundro, 1991]. The modular construction of ICSIM also allows an easy portation to dedicated hardware, eg. the neural network coprocessor RAP [Morgan, 1990].

8.2 Unification

The first step in the implementation of CHCL was a realization of the unification algorithm [Kurfeß, 1991a]. The structure and behavior of the unification network are derived in a straightforward way from the algorithm described in Section 4, or in more detail in [Hölldobler, 1990a].

Structure of the Network The unification network is modeled according to the representation of terms as position-label pairs. These pairs are arranged as a matrix, with positions as rows and symbols as columns. Each entry of the matrix consists of one *term unit* and as many *unification units* as there are positions in the unification problem to be evaluated. The term unit indicates if a symbol occurs at a certain position, or, during the unification process, if it has the same value as another symbol at the same position. The unification units are used to indicate that the corresponding term unit has to be activated during the unification process.[3] Both term and unification units are simple boolean units with a threshold activation function; the output is 0 if the weighted sum of the inputs is lower than the threshold, and 1 if the sum is equal to or greater than the threshold. There are two types of *links* in the network: *weak links* with a weight $w = 1$, and *strong links*, with a weight w equal to the sum of possible weak connections for any unit in the network, which is $\mathtt{w} = \frac{1}{2} \times \mathtt{n} \times \mathtt{m} \times (\mathtt{m} - \mathtt{1})$, where n is the number of symbols and m is the number of positions in a unification problem.

A term unit has $\mathtt{2} \times (\mathtt{m} - \mathtt{1})$ strong inputs: from each of its own unification units except for the one pointing to its own position, and from the unification unit corresponding to its own position of the other occurrences of the same symbol.[4] The threshold of a term unit $\mathtt{t_t} = \frac{1}{2} \times \mathtt{n} \times \mathtt{m} \times (\mathtt{m} - \mathtt{1})$ is the same as the weight of a strong connection, so that the term unit is activated as soon as one of its unification units raises its output. The threshold of a unification unit $\mathtt{t_u} = \mathtt{t_t} + \mathtt{1}$ is slightly higher than that of a term unit, and, hence, the unification unit is only activated if at least one strong link plus one weak link is active on its inputs. A symbol s at the first position, for example, receives inputs form all its own unification units except for the first one, and from all the first unification units of the occurrences of the same symbol at other positions. Within one position, there are weak links between unification units corresponding to the same position; in ICSIM terminology, these units are bus-connected. Here we have to distinguish between two types of symbols: variables on one hand, and function symbols and constants on the other. The weak links between the unification units within a position go from each variable to all other symbols; this mirrors the effect of shared variables, namely the occurrence of the same variable symbol at different positions. The links described so far are used to achieve SINGULARITY; all instances of a symbol at different positions must have the same value, and all symbols occurring at the same position must also have the same value. Figure 5 gives an overview of these singularity links in the network. The links between term units and unification units of other positions are shown for one symbol only on the right side of the picture.

Additional links are required to achieve the other operation in the unification algorithm, DECOMPOSITION. This operation performs the unification of the corresponding arguments $\mathtt{s_i}$ and $\mathtt{t_i}$, $\mathtt{1} \leq \mathtt{i} \leq \mathtt{n}$, of two subterms $\mathtt{f}(\mathtt{s_1}, \ldots, \mathtt{s_n})$ and $\mathtt{f}(\mathtt{t_1}, \ldots, \mathtt{t_n})$. Decomposition is relevant if two subterms are forced to assume the same value due to shared variables, which in our example is the case for g(a) and g(Y) because of the occurrence of the variable X at both of these positions in the other term. As a consequence, the corresponding arguments must assume the same value, in our case a and Y. The propagation of decomposition from two parent symbols to their respective children requires weak links from the unification unit of the one parent position which connects it to its partner parent position to the respective unification units of all symbols at the particular child positions. Figure 6 shows the decomposition links required in our example. The variable X occurs at positions 1 and 3, and weak links must be established from the fifth unification unit of position 1 (which represents the relation to the partner at position 3) to the sixth unification unit of its child at position 1.1, and from unification unit 2 of position

[3]In the current implementation, the network is compiled from scratch for each new unification problem; a generic version which adapts a precompiled network to the particular problem is under construction. The current implementation also uses twice as many unification units as necessary, which made the implementation a little easier and does not do any harm.

[4]One set of unification units actually would be sufficient.

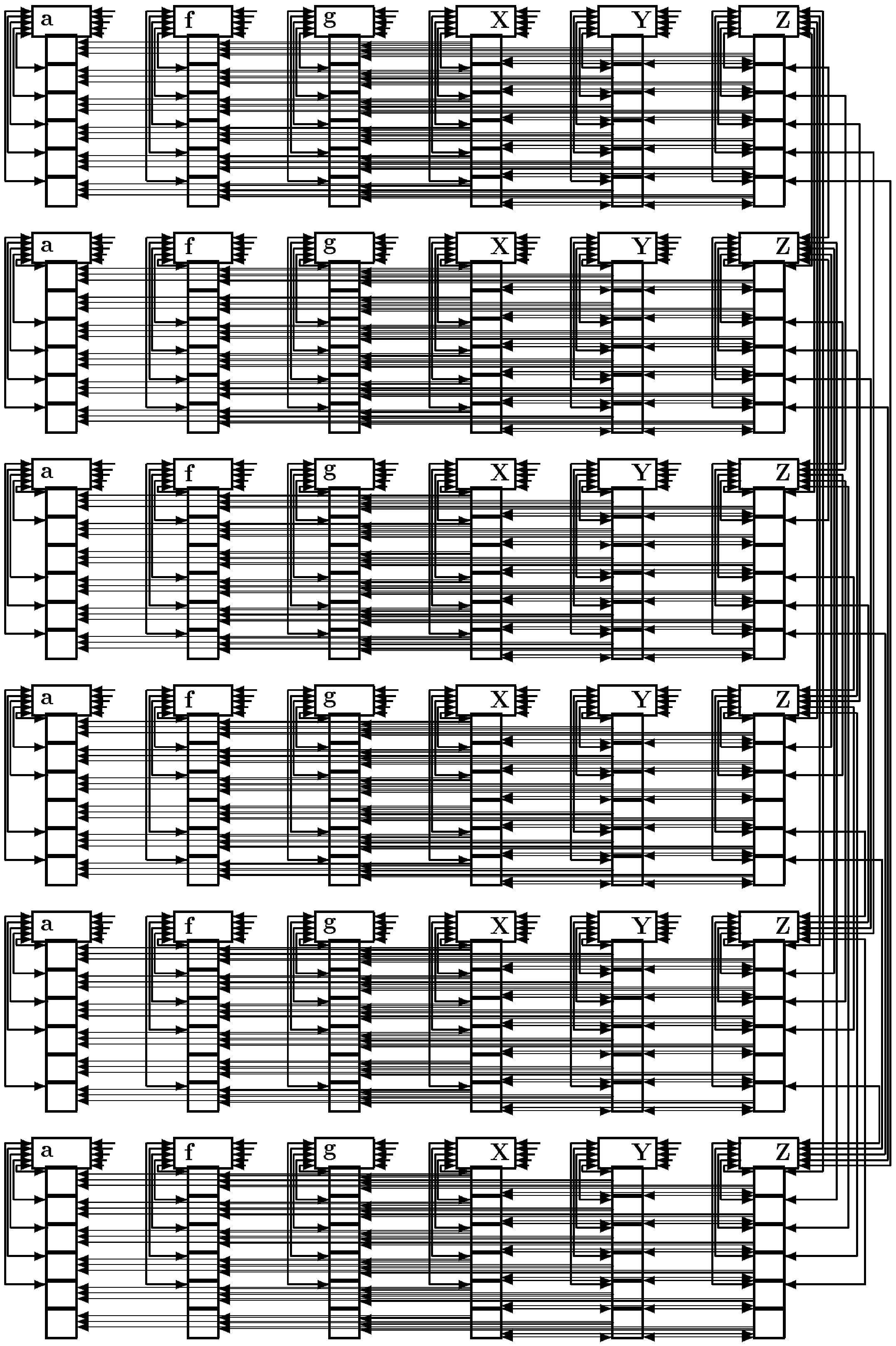

Figure 5: Unification network with singularity links.

3 to the third unification unit of position 3.1. These decomposition links are established at compile time; they must be installed from all variables of parent positions to all symbols at the respective children positions since shared variables might appear during the execution of the unification algorithm. Decomposition links only emanate from variables because the occurrence of two different constants or function symbols at the same position would make the two terms incompatible, and unification fails.

The size of the network in this implementation is of the order of $\mathcal{O}(\mathtt{m} \times \mathtt{n}^2)$, where n is the number of positions and m the number of symbols; the number of connections is of the order of $\mathcal{O}(\mathtt{m}^2 \times \mathtt{p}^2)$.

Execution of the Algorithm The effects of the computation of the unification algorithm in the network are summerized in a trace of our example, shown in Figure 7. One step in the execution is one sweep through all the units in the network; whereas ICSIM allows for various kinds of asynchronicity in the execution[5] we assume synchronous evaluation here for the sake of easier understanding.

The information within a single step is arranged as a matrix of symbols and positions, the symbols in the columns and the positions in the rows. A symbol consists of a term unit (represented by "=" or "#"), and a number of unification units (represented by "-" or "|"). The occurence of a symbol s at a position π corresponds to an active term unit at the appropriate place in the matrix and is shown as "#"; an active unification unit "|") indicates that one or both of the partner symbols identified by this unification unit is active.

Initially, only the symbol units corresponding to the occurrences in the two terms are active, eg. f at position 0, g and X at position 0.1, etc. In the second step, a number of unification units gets active because they receive inputs from the two partner term units they connect. The third step shows the activation of some more unification units, this time via weak links. Consider for example the second unification unit of symbol g at position 0.2; it receives a strong input from the term unit of the same symbol at position 0.1 and a weak input from the second unification unit of symbol X at position 0.2, which has been active in the previous step. This is an effect of the singularity connections, which also leads to the activation of more unification units for the symbols g and Y. The two active unification units for the symbols a and Z are due to the decomposition links: the third unification unit of a at position 0.3.1 receives a strong input from the active term unit of a at the third position, 0.1.1, and weak input from the second unification unit of the symbol X at position 0.3, which is the parent postion of 0.3.1. In a similar way the last unification unit of Z at 0.1.1 is activated due to a strong input from the term unit Z at 0.3.1 and a weak input from the fifth unification unit of X at its parent position 0.1. In step 4, term units of the symbols a, g, Y, and Z are activated as a consequence of the activation of the unification units form the previous step. In the last step, some more unification units become active, but without further consequences, and the network has reached a fixpoint.

[5]Eg. not all units are evaluated in one sweep.

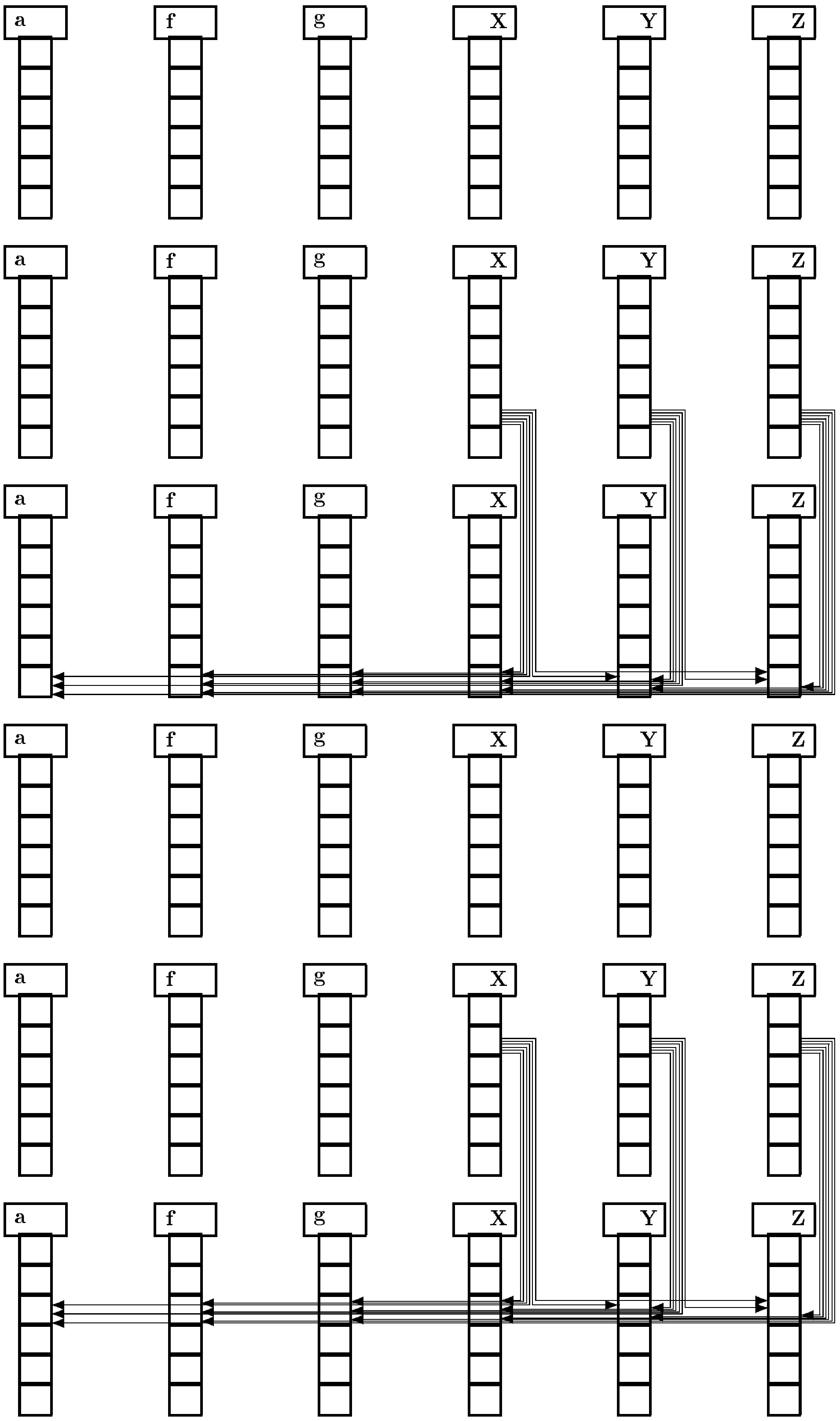

Figure 6: Unification network with decomposition links.

```
STEP: 1
  a         f         g             X         Y         Z
= ------  # ------  = ------      = ------  = ------  = ------  0
= ------  = ------  # ------      # ------  = ------  = ------  0.1
# ------  = ------  = ------      = ------  = ------  = ------  0.1.1
= ------  = ------  = ------      # ------  # ------  = ------  0.2
= ------  = ------  # ------      # ------  = ------  = ------  0.3
= ------  = ------  = ------      = ------  = ------  # ------  0.3.1
STEP: 2
  a         f         g             X         Y         Z
= ------  # ------  = ------      = ------  = ------  = ------  0
= ------  = ------  # ----|-      # ---||-  = ------  = ------  0.1
# ------  = ------  = ------      = ------  = ------  = ------  0.1.1
= ------  = ------  = ------      # -|--|-  # ------  = ------  0.2
= ------  = ------  # -|----      # -|-|--  = ------  = ------  0.3
= ------  = ------  = ------      = ------  = ------  # ------  0.3.1
STEP: 3
  a         f         g             X         Y         Z
= ------  # ------  = ------      = ------  = ------  = ------  0
= ------  = ------  # ---||-      # ---||-  = ---|--  = ------  0.1
# ------  = ------  = ------      = ------  = ------  = -----|  0.1.1
= ------  = ------  = -|--|-      # -|--|-  # -|--|-  = ------  0.2
= ------  = ------  # -|-|--      # -|-|--  = ---|--  = ------  0.3
= --|---  = ------  = ------      = ------  = ------  # ------  0.3.1
STEP: 4
  a         f         g             X         Y         Z
= ------  # ------  = ------      = ------  = ------  = ------  0
= ------  = ------  # ---||-      # ---||-  # ---||-  = ------  0.1
# ------  = ------  = ------      = ------  = ------  # -----|  0.1.1
= ------  = ------  # -|--|-      # -|--|-  # -|--|-  = ------  0.2
= ------  = ------  # -|-|--      # -|-|--  # -|-|--  = ------  0.3
# --|---  = ------  = ------      = ------  = ------  # ------  0.3.1
STEP: 5
  a         f         g             X         Y         Z
= ------  # ------  = ------      = ------  = ------  = ------  0
= ------  = ------  # ---||-      # ---||-  # ---||-  = ------  0.1
# -----|  = ------  = ------      = ------  = ------  # -----|  0.1.1
= ------  = ------  # -|--|-      # -|--|-  # -|--|-  = ------  0.2
= ------  = ------  # -|-|--      # -|-|--  # -|-|--  = ------  0.3
# --|---  = ------  = ------      = ------  = ------  # --|---  0.3.1
Fixpoint reached...
```

Figure 7: Trace of the computation performed in an attempt to unify `f(X,X,X)` and `f(g(a),Y,g(Z))`.

8.3 CHCL

In the last subsection we have given a detailed description of the implementation of the term and unification layer and of the operations performed on these layers. The remaining layers can easily be implemented in very much the same way. The units and links are precisely specified in [Hölldobler, 1990a].

9 Future Work

The current status of CHCL shows the correctness of the approach through its formal description as well as its feasibility through its implementation. There is no doubt, however, that in order to be useful for a wider range of applications CHCL must be enhanced and improved with respect to a number of issues. One of the main points is to enhance the functionality by allowing copies of clauses, and providing means to deal with cycles and recursion. Another is the selection of *promising* spanning matings, eg. through strategies based on learning from previous examples. The implementation needs an increase in efficiency, which may be achieved by mechanisms to restrict the execution to those units which are actually needed for the current program. These issues are described in more detail below.

9.1 Functionality

In its current version, CHCL is restricted by the requirement that only one instance of a clause may be used, and there are no copies. Hence, the logic is decidable and a large and important class of formulae cannot be handled. This is an instance of a more general problem encountered in almost all connectionist systems developed so far. It is the problem of how to recruit new units if the units used so far are incapable of representing additional knowledge. Techniques like recruitment learning [Feldman, 1982; Diederich, 1988] are promising, but the structures recruited so far using this technique are much simpler than the structures used in CHCL.

Copies An extension to handling a finite number of clause copies can be based on the use of different phases in the execution for the different copies as described in [Shastri and Ajjanagadde, 1990a]. It is not clear, however, if this approach can handle an arbitrary number of clauses at runtime, which would be necessary in the case of recursive formulae. In addition, one may argue that this is a clever implementation maneuver, but not really a fundamental solution to the problem how to represent multiple instances of a clause. Another possibility is the use of an recursive auto-associative memory [Pollack, 1988; Pollack, 1990], which provides a way to store variable-sized recursive data structures in patterns of a fixed width.

Cycles Cycles in a logical formula can be handled in two ways: the first is to explicitly resolve them by creating as many new copies of the affected clauses as necessary; the other is to analyze the structure of the formula, detect cycles, and apply transformations to the formula to eliminate or simplify the cycles. The latter basically is an analysis of the graph given by the logical connections of the formula, and to some degree can be applied statically at compile time [Bibel *et al.*, 1991]. An approach to improve the first one can be based on observing the execution of the formula: if certain actions, eg. the creation of clause copies, are repeated over and over, it might be possible to identify the changes for each iteration, and modify the execution scheme accordingly.

Spanning Matings The treatment of spanning matings currently is sequential: one is investigated after the other. The other extreme would be to create a copy of the network for each spanning set and examine the unifiability in parallel.

9.2 Learning

Strategies The selection of the next spanning set is currently based on syntactic criteria. This can be enhanced by learning semantic strategies based on experience gathered from the solution process of previously solved formulae [Suttner, 1989; Schumann *et al.*, 1989; Ultsch *et al.*, 1990]. Learning can be useful in different stages of the execution, and on different levels in the network. It can be applied during the detection and elimination or transformation of cycles, for the selection of promising spanning sets, and for the creation of clause copies. It is questionable, however, if general strategies can be derived from the structure of the formula alone; more likely is the development of strategies for certain domains, or classes of applications.

9.3 Implementation

The completion of the current implementation is expected for the summer of 1991. This implementation certainly will not rival logic programming systems or theorem provers with respect to speed and functionality; it is meant to demonstrate the feasibility of the approach, and for experimentation. Mid-term goals are the portation to dedicated connectionist architectures like ICSI's RAP [Morgan, 1990], or to massively parallel systems like the connection machine [Thi, 1987].

Representation An important goal for future implementations will be to overcome the quadratic number of units. In most cases, only a small fraction of the units are actually used for a particular problem; thus the recruitment of units from an overall pool of units and the dynamical establishment of the necessary links is a promising way to substantially reduce the space requirements. Another problem concerns the representation of clause copies: Currently only one copy of a clause is allowed, which certainly is an intolerable restriction in the long run. As mentioned before, possible solutions for this problem are the use of phases, indices or signatures or the use of a recursive auto-associative memory. Promising first experiments using a recursive auto-associative memory to learn a restricted version of the unification problem have been performed by [Stolcke and Wu, 1991].

Execution As a result of the enormous number of units and connections, execution times at the moment are rather slow even on relatively fast workstations. In addition to the use of dedicated or massively parallel machinery, optimizations are pursued in the implementation of

Icsim . Virtual units or working memory can be used to achieve a lazy evaluation scheme, only evaluating units whose inputs have changed. In the Chcl implementation, garbage collection could eliminate rows and columns in the term layer which become identical during evaluation. Another possiblity is the pre-compilation of the body of a logic program (without the query), adding the necessary units for a particular query upon request; this approach, however, mainly decreases compilation, not execution time. Although unification is pursued in a parallel way, it still may consume a considerable part of the execution time. Here weak unification may be useful to eliminate spanning sets before full unification is applied, or which are not unifiable for any query.

9.4 Applications

The ultimate goal of Chcl is not only its implementation, but also the usage for *real* applications. One currently pursued is spatial reasoning in the context of the L_0 project at the International Computer Science Institute [Feldman *et al.*, 1990]. The intention of this project is to associate sentences describing spatial relations between objects and the corresponding scenes. Chcl is particularly suited for such a task: sentences can be represented as logical formulae, whereas the representation of scenes can be based on a connectionist representation.

Acknowledgement Most of work on Chcl has been done while both authors were at the International Computer Science Institute. We are therefore indebted to Jerry Feldman for providing such an exciting and stimulating environment which made this research possible. We also like to thank Andreas Stolcke, Susan Hollbach-Weber, Heinz W. Schmidt, Lokendra Shastri, Joachim Diederich, Hans-Werner Güsgen, and Walter Hower for their constructive criticism and their help in many ways.

References

[Aho and Ullman, 1972] A. V. Aho and J. D. Ullman. *The Theory of Parsing, Translation, and Compiling*, volume I: Parsing. Prentice-Hall, Englewood Cliffs, N. J., 1972.

[Ballard, 1986] D. H. Ballard. Parallel logic inference and energy minimization. In *Proceedings of the AAAI National Conference on Artificial Intelligence*, pages 203 – 208, 1986.

[Bibel *et al.*, 1991] W. Bibel, S. Hölldobler, and J. Würtz. Cycle unification. Technical Report AIDA-91-15, FG Intellektik, FB Informatik, TH Darmstadt, 1991.

[Bibel, 1987] W. Bibel. *Automated Theorem Proving.* Vieweg Verlag, Braunschweig, second edition, 1987.

[Bibel, 1988] W. Bibel. Advanced topics in automated deduction. In Nossum, editor, *Fundamentals of Artificial Intelligence II.* Springer, 1988.

[Colmerauer, 1982] A. Colmerauer. Prolog and infinite trees. In Clark and Tarnlund, editors, *Logic Programming*, pages 231–251. Academic Press, 1982.

[Cooper and Swain, 1988] P. R. Cooper and M. J. Swain. Parallelism and domain dependence in constraint satisfaction. Technical Report 255, Computer Science Department, Univ. of Rochester, 1988.

[Diederich, 1988] J. Diederich. Connectionist recruitment learning. In *Proceedings of the European Conference on Artificial Intelligence*, pages 351–356, 1988.

[Dowling and Gallier, 1984] W. F. Dowling and J. H. Gallier. Linear-time algorithms for testing the satisfiability of propositional Horn formulae. *Journal of Logic Programming*, 1984.

[Dwork *et al.*, 1984] C. Dwork, P. C. Kannelakis, and J. C. Mitchell. On the sequential nature of unification. *Journal of Logic Programming*, 1:35–50, 1984.

[Elman, 1989] J. L. Elman. Structured representations and connectionist models. In *Proceedings of the Annual Conference of the Cognitive Science Society*, pages 17–25, 1989.

[Feldman and Ballard, 1982] J. A. Feldman and D. H. Ballard. Connectionist models and their properties. *Cognitive Science*, 6(3):205–254, 1982.

[Feldman *et al.*, 1990] J. A. Feldman, G. Lakoff, A. Stolcke, and S. H. Weber. Miniature language acquisition: A touchstone for cognitive science. In *Proceedings of the Annual Conference of the Cognitive Science Society*, pages 686–693, 1990.

[Feldman, 1982] J. A. Feldman. Dynamic connections in neural networks. *Biological Cybernetics*, 46:27–39, 1982.

[Güsgen and Hölldobler, 1991] H.-W. Güsgen and S. Hölldobler. Connectionist inference systems. In B. Fronhöfer and G. Wrightson, editors, *Parallelization in Inference Systems.* Springer, 1991. (to appear).

[Güsgen, 1990] H. W. Güsgen. A connectionist approach to symbolic constraint satisfaction. Technical Report TR-90-018, International Computer Science Institute, Berkeley, CA, 1990.

[Hölldobler and Hower, 1991] S. Hölldobler and W. Hower. Constraint satisfaction in a connectionist inference system. In *Proceedings of the International Symposium on Artificial Intelligence*, 1991. (to appear).

[Hölldobler, 1990a] S. Hölldobler. CHCL - A connectionist inference system for a limited class of Horn clauses based on the connection method. Technical Report TR-90-042, International Computer Science Institute, Berkeley, CA, 1990.

[Hölldobler, 1990b] S. Hölldobler. A structured connectionist unification algorithm. In *Proceedings of the AAAI National Conference on Artificial Intelligence*, pages 587–593, 1990. A long version appeared as Technical Report TR-90-012, International Computer Science Institute, Berkeley, California.

[Jordan, 1986] M. I. Jordan. Attractor dynamics and parallelism in a connectionist sequential machine. In *Proceedings of the Annual Conference of the Cognitive Science Society*, 1986.

[Kurfeß, 1991a] F. Kurfeß. Unification on a connectionist simulator. In *Proceedings of the International Conference on Artificial Neural Networks*, 1991. ICSI.

[Kurfeß, 1991b] F. Kurfeß. Unification with icsim. Technical report, International Computer Science Institute, Berkeley, CA, 1991.

[Mackworth, 1987] A. Mackworth. Constraint satisfaction. In Shapiro, editor, *Encyclopedia of Artificial Intelligence*, pages 205–211. John Wiley & Sons, 1987.

[Meyer, 1988] B. Meyer. *Object-Oriented Software Construction.* Prentice Hall, 1988.

[Morgan, 1990] N. Morgan. The ring array processor (RAP): Algorithms and architecture. Technical Report TR-90-47, International Computer Science Institute, Berkeley, CA, 1990.

[Omohundro, 1991] S. Omohundro. Differences between Sather and Eiffel. *Eiffel Outlook*, 1991.

[Paterson and Wegman, 1978] M. S. Paterson and M. N. Wegman. Linear unification. *Journal of Computer and System Sciences*, 16:158–167, 1978.

[Pollack, 1988] J. B. Pollack. Recursive auto-associative memory: Devising compositional distributed representations. In *Proceedings of the Annual Conference of the Cognitive Science Society*, 1988.

[Pollack, 1990] J. B. Pollack. Recursive distributed representations. *Artificial Intelligence*, 46:77–105, 1990.

[Robinson, 1965] J. A. Robinson. A machine-oriented logic based on the resolution principle. *Journal of the ACM*, 12:23–41, 1965.

[Schmidt, 1990] H. W. Schmidt. ICSIM: Initial design of an object-oriented net simulator. Technical Report TR-90-055, International Computer Science Institute, Berkeley, CA, 1990.

[Schumann *et al.*, 1989] J. Schumann, W. Ertel, and C. Suttner. Learning heuristics for a theorem prover using back propagation. In *Österreichische AI Tagung*, 1989.

[Scutella, 1990] M. G. Scutella. A note on Dowling and Gallier's top-down algorithm for propositional Horn satisfiability. *Journal of Logic Programming*, 8(265-273), 1990.

[Shastri and Ajjanagadde, 1990a] L. Shastri and V. Ajjanagadde. From associations to systematic reasoning: A connectionist representation of rules, variables and dynamic bindings. Technical Report MS-CIS-90-05, Department of Computer and Information Science, University of Pennsylvania, Philadelphia, School of Engineering and Applied Science, PA 19104-6389, 1990.

[Shastri and Ajjanagadde, 1990b] L. Shastri and V. Ajjanagadde. An optimally efficient limited inference system. In *Proceedings of the AAAI National Conference on Artificial Intelligence*, pages 563–570, 1990.

[Smolensky, 1987] P. Smolensky. On variable binding and the representation of symbolic structures in connectionist systems. Technical Report CU-CS-355-87, Department of Computer Science & Institute of Cognitive Science, University of Colorado, Boulder, CO 80309-0430, 1987.

[Stickel, 1987] M. E. Stickel. An introduction to automated deduction. In W. Bibel and P. Jorrand, editors, *Fundamentals of Artificial Intelligence*, pages 75 – 132. Springer, 1987.

[Stolcke and Wu, 1991] A. Stolcke and D. Wu. Tree matching with recursive distributed representations. International Computer Science Institute, Berkeley, CA, 1991. submitted to NIPS 91.

[Suttner, 1989] C. Suttner. Learning heuristics for automated theorem proving. Master's thesis, Institut für Informatik, Technische Universität München, 1989.

[Thi, 1987] Thinking Machines Corporation. Connection Machine Model CM-2 Technical Summary. Technical Report HA87-4, 1987.

[Touretzky and Hinton, 1988] D. S. Touretzky and G. E. Hinton. A distributed connectionist production system. *Cognitive Science*, 12:423 – 466, 1988.

[Ultsch *et al.*, 1990] A. Ultsch, R. Hannuschka, U. Hartmann, and V. Weber. Learning of control knowledge for symbolic proofs with backpropagation networks. In R. Eckmiller, G. Hartmann, and G. Hauske, editors, *Parallel Processing in Neural Systems and Computers*, pages 499–502. Elsevier, 1990.

[Weber and Stolcke, 1990] S. H. Weber and A. Stolcke. L_0: A testbed for miniature language aquisition. Technical Report TR-90-010, International Computer Science Institute, Berkeley, CA, 1990.