Unification with ICSIM

Franz Kurfeß

TR-91-053

August 1991

Abstract

This document describes the implementation of a distributed unification algorithm using the connectionist simulator ICSIM. The algorithm is based on S. Hölldobler's work, as described in [Hölldobler, 1990b]. Unification problems are specified according to a simple language, describing the terms, functions, variables and constants occuring in such a problem; the terms to be unified are represented as $<$ `term_1` $=$ `term_2` $>$ (e.g. $< f(x,x,x) = f(g(a),y,g(z)) >$).

A parser extracts relevant information and creates intermediate data structures needed for the construction of the connectionist network. Essential data structures describe the symbols occurring in the terms, the hierarchical structure of the terms (functions and their arguments), and the occurrences of the symbols in the terms. The connectionist unification network is constructed based on these intermediate structures. It is hierarchically organized, its top level `NET` consisting of `POSITIONS`, which correspond to the nodes in the term structure. A `POSITION` consists of a `SYMBOL`, either of type `VARIABLE` or `CONSTANT`. Symbols comprise a `TERM UNIT` and a number of `UNIFICATION UNITS`, depending on the number of positions in the terms to be unified. Initially, `TERM UNITS` are set according to the occurrences of their symbols in the term structure; based on the links within the network and the activation of `UNIFICATION UNITS`, more `TERM UNITS` are activated as required by the unification algorithm. The final set of active `TERM UNITS` is used to construct the most general unifier for the terms to be unified. The network can be easily extended to detect inconsistencies in the term structure or to perform an occur check.

1 An Overview of the Unification System

Unification serves as basic operation in a number of symbolic processing formalisms. It is used to find a match between a set of data structures, usually represented as terms. Most implementations of unification rely heavily on the manipulation of pointers and assignment of values to variables, two tasks at which –at least for the time being – connectionist models are not very good. The approach described here is based on occurrence - label pairs, where the presence of a symbol at a certain position is indicated by an entry in a matrix with the symbols and positions as rows and columns, respectively. This matrix serves as a skeleton for a connectionist network, augmented by links and additional units according to the structure of the terms involved. A spreading activation scheme then performs operations on the network which constitute unification.

1.1 Representation of the Unification Problem

The unification problem is presented to the system as a pair of two terms, separated by the equality symbol and surrounded by left and right angles. As an example, consider the task of unifying the two terms $f(x, x, x)$ and $f(g(a), y, g(z))$, which are represented as $< f(x, x, x) = f(g(a), y, g(z)) >$.[1] In order to be unifiable, the structure of the two terms must be compatible, and thus both terms can be described as the nodes of one tree. This tree can be viewed as the "overlay" of the trees corresponding to each term, and its nodes may be labeled by one or two components, one from each of the two terms. The above example then looks as follows (the vertical bar separates the components of the two terms):

$$\begin{array}{ccccc} & & f \mid f & & \\ x \mid g & & x \mid y & & x \mid g \\ \mid a & & & & \mid z \end{array}$$

This representation scheme is used to incorporate the essential information of the unification problem into the connectionist network which is used to solve the problem. The nodes of the tree will be referred to as *positions*, numbered in a depth-first, left-to-right manner; for our example, the tree consists of positions

$$\begin{array}{ccccc} & & \mathbf{0} & & \\ \mathbf{0.1} & & \mathbf{0.2} & & \mathbf{0.3} \\ \mathbf{0.1.1} & & & & \mathbf{0.3.1} \end{array}$$

[1] This method can easily be extended to *set unification*, unifying not only a pair but a set of terms. For the sake of simplicity, we restrict ourselves to the unification of term pairs.

A position may be labeled by one or two *symbols*. A symbol may be of type *variable* or *constant*; the latter includes function symbols.[2] The subterm relation induced by the arguments of functions is represented by explicit connections in the network.

In the unification network, a position is encoded as a vector of units, one for each different symbol occuring in the unification problem. If a symbol occurs at a certain position, the corresponding unit is activated; the network of position vectors for our example is depicted in Figure 1.

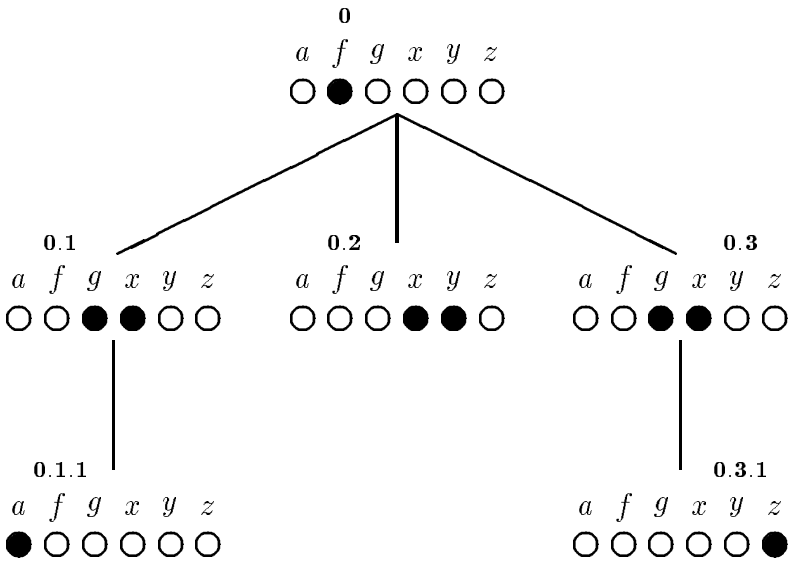

Figure 1: Unification network: Initial state

The picture shown in Figure 1 describes the initial state of the network for the unification problem $< f(x,x,x) = f(g(a),y,g(z)) >$. In the next section we will examine which actions have to be performed in the network in order to unify the two terms.

1.2 Evaluation of the Unification Problem

Without going into the details the underlying distributed unification algorithm, two essential operations are performed during the unification process. The first one is to guarantee that all instances of a particular symbol in the pair of terms have the same value. For constants, this is trivial; for variables it means that all the substitutions for one variable must be compatible. This feature of unification will be referred to *singularity* [Hölldobler, 1990b]. Looking at our example we can identify three positions where singularity is relevant: Position **0.1** shares the constant g with position **0.3** and the variable x with position **0.2**, which in turn shares the variable y with

[2] As a convention, symbols are identified by their designators, typically $a, b, c, \ldots$ for constants, $f, g, h, \ldots$, for function symbols, and s, t, u, v, w, x, y, z for variables; if necessary, indices are used, such as $x1, x2, x3, \ldots$.

position **0.3**. Thus x, y and g must all have the same value, which in this case is determined by the constant g; actually g stands for the name of a function here, and the resulting value is determined by the evaluation of this function. The required effect of singularity on our unification network is shown in Figure 2. The sharing of values is expressed by the fact that in the three positions **0.1**, **0.2**, and **0.3** the same set of symbol units is activated, namely g, x, and y (the newly activated symbol units are marked by only partially filled circles).

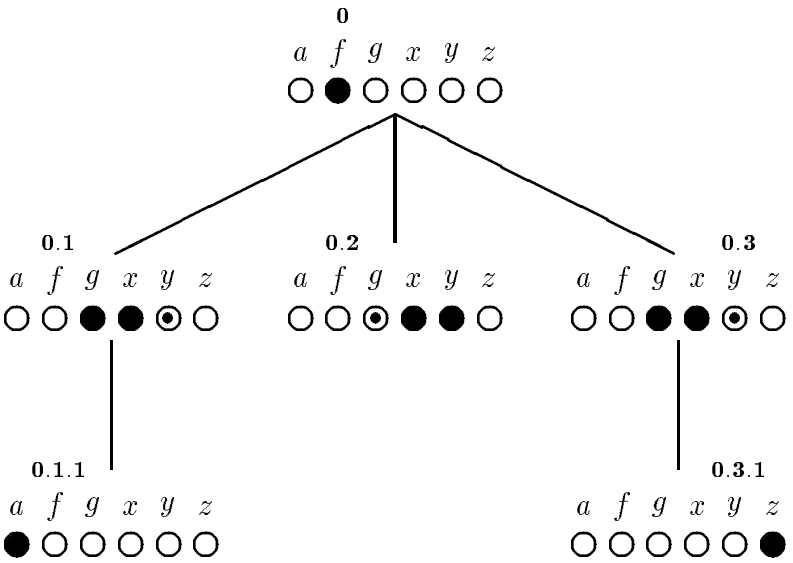

Figure 2: Unification network: Singularity

The second operation is the "forwarding" of the unification task through the arguments of functions; it requires that if two positions are unified through a shared variable, their respective arguments (if there are any) must also be unified pairwise. This feature is called *decomposition*. In our examples it occurs at positions **0.1.1** and **0.3.1**: Their predecessors, positions **0.1** and **0.3** have been unified through singularity. The required effect is the same as with singularity, the affected positions must have the same set of activated symbol units. The final state of the unification network is shown in Figure 3.

Please note that another important aspect of unification, the compatibility of the structure of the terms involved (homogeneity) is not taken into account here. It must either have been checked already before the evaluation of the network (e.g. in the parsing phase), or the network must be extended for homogeneity.

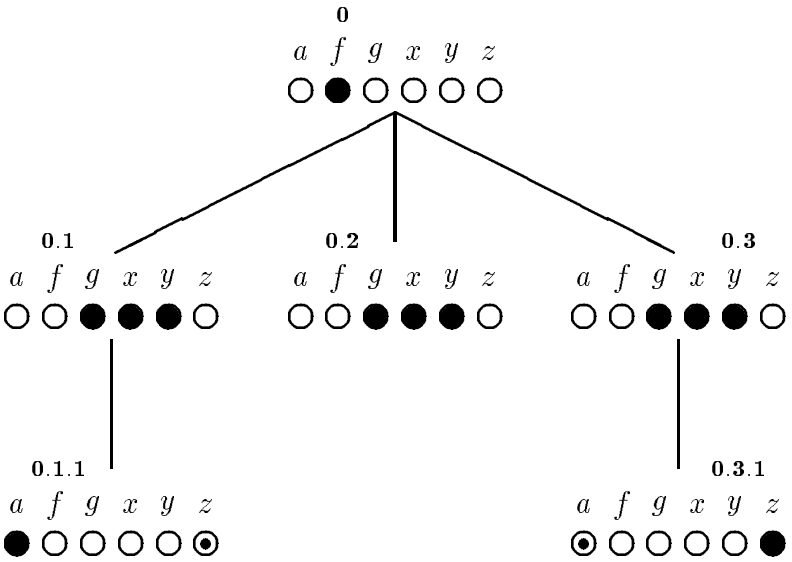

Figure 3: Unification network: Decomposition

2 The Connectionist Simulator ICSIM

ICSIM is a simulator for connectionist networks under development at ICSI [Schmidt, 1990]. It provides a collection of basic modules for the construction of connectionist networks, while maintaining a high degree of flexibility through the use of an object-oriented implementation environment. A first version of ICSIM was developed in EIFFEL [Meyer, 1988], a second one in SATHER[Omohundro, 1990, Omohundro, 1991], a derivative of EIFFEL geared towards higher efficiency and more simplicity.

ICSIM relieves the user from the tedious task of implementing all the necessary details down to the level of the single units, their interconnections, and their possible modes of operation and interaction. Its goal is to let the user concentrate on the *system level* instead, putting more emphasis on investigation and experimentation with novel applications and / or artificial neural network concepts.

2.1 Relevant Classes for the Unification System

This implementation of a network for unification only uses a small fraction of the classes available in ICSIM. These classes are described below.

2.1.1 Views

ICSIM provides a collection of classes for interaction and the visual representation of networks to the user. In its present version, only character-oriented output is possible, and interaction is done through simple menus. A graphical interface is under development. For the unification system, basically only one class, `TEXT_VIEW_1NET`, is used to provide textual command interaction with the network.

2.1.2 Nets

The whole unification system consists of a hierarchy of networks. It is based on the class `HIERARC_NET`, which allows the composition of one-dimensional nets of nets. The components of a net again can be nets, leading to hierarchically nested networks. The lowest level of the hierarchy usually consist of units.

2.1.3 Units

Both the symbol and the unification units are implemented via the class `BOOL_UNIT`, which compares the weighted sum of its inputs to a threshold, and delivers `true` or `false` as result.

2.1.4 Connections

ICSIM offers a set of pre-defined operations to establish connections between components of a network. `complete-connect` builds a completely connected network

with connections from each component to all others. `cross-connect` operates on two networks and establishes connections from each component of one network to all components of the other. `bus-connect` connects a component of one network to the corresponding component of the other network, resulting in "parallel" connections between the two networks.

For the connections of the unification network, these basic connection mechanisms were not sufficient, however, and specialized operations had to be defined. This is particularly due to the necessity of different types of interconnection schemes (e.g. for singularity, decomposition, within a position, within a symbol).

3 The Unification Network

3.1 Hierarchical Structure

The unification network as a whole consists of a hierarchically layered network. The top level is the net itself, described by the class `NET`. It is composed of instances of the class `POSITION`; the number of instances is determined by the term pair to be unified. The relation between positions, describing the fact that one occurs as an argument of another, is not directly visible on this level; it is expressed explicitly through connections on lower levels (see Section 3.6). Each of the positions contains a vector of instances of the class `SYMBOL`, grouped into the two types `CONSTANT` and `VARIABLE`. The vector contains all symbols occuring in the unification problem. A symbol is composed of `UNIT`s: one `SYMBOL UNIT` indicates if the symbol is activated or not, and `UNIFICATION UNIT`s establishing connections between symbols at different positions. Since a symbol has to be able to differentiate between connections emanating from different positions, the number of unification units equals the number of positions.[3]

In the following we describe the components of the network in more detail: first the connections, then the fundamental units and the connections between them, and finally how they are put together to construct the higher layers.

3.2 Connections

The unification network requires two types of connections: strong connections and weak connections, both associated with an integer number as their weight w. Weak connections have the weight $w = 1$, whereas the weight associated with strong connections depends on the structure of the unification problem. The weight in this case must be greater than the sum of weights of all possible weak connections into one unit; here it is $w = \frac{1}{2} \times n \times m \times (m - 1)$, n being the number of positions in the unification problem, and m the number of constant and variable symbols occuring in the problem.

3.3 Units

The basic units in the unification network are boolean units. Their potential is computed as the weighted sum of the inputs, which is compared against the threshold. If the potential is lower than the threshold the output is 0 (or `false`), otherwise 1 (or `true`). The thresholds for symbol and unification units are defined dependent on their inputs and on which combination of inputs is supposed to activate the unit.

[3] Actually one less than the number of positions would be sufficient since each symbol is part of one particular position; the construction of the network, however, is simpler when neglecting this fact.

3.3.1 Symbol Units and Their Connections

Each symbol component of the network contains a single symbol unit which indicates the state of the symbol. A symbol unit is initially activated if the symbol occurs at that particular position; in our example, f is activated in position **0**, g in positions **0.1** and **0.3**, and so on. A symbol unit may also become activated during the evaluation of the network. This is the case when the weighted sum of its inputs reaches the threshold. The inputs of a certain symbol unit come from all the unification units of the same symbol, and from one unification unit of each other position. They are of the strong type; the corresponding connections are bi-directional so that the same unification units also receive inputs from this symbol unit. The threshold of a symbol unit is set to the value $t = \frac{1}{2} \times n \times m \times (m-1)$; this is exactly the same as the weight of a strong connection. Thus a symbol unit gets activated as soon as at least one of its unification units has been activated.

3.3.2 Unification Units and Their Connections

The purpose of the unification units is to indicate if the corresponding symbol unit at a particular position has to be activated due to unification actions the symbol becomes involved in. A unification unit has strong bi-directional connections to two instances of the same symbol at different positions. This reflects the request of singularity, namely to guarantee that all the instances of a symbol at different positions have the same value. The other aspect of singularity, namely that different symbols appearing at the same position must have the same value, is also achieved through unification units. Within one position, each unification unit of a particular symbol is weakly connected to the corresponding unification unit of all the different symbols. The details of these connections will be discussed on the position level in Section 3.5. In addition to these connections dealing with singularity, a unification unit may also have connections induced by decomposition. These connections are of the weak type, and are directed from a unification unit of a position which corresponds to a function to a unification unit in an argument position; a more detailed description of the interconnection structure for decomposition follows in Section 3.6.

A unification unit is activated only if at least one of its two symbol units has already been activated and at least one other connection comes from an activated unit. If this other unit is the second symbol unit connected with the unification unit under consideration, the latter will be activated, but without direct implications. If the other unit is a unification unit connected as an outcome of singularity or decomposition our unification unit gets activated and in turn also activates its second symbol unit.

The threshold of a unification unit is set to $w+1$, where w is the weight associated with a strong connection. This mirrors the fact that a unification unit is activated only if at least one strong connection together with another (strong or weak) connection is active.

3.4 Symbols

Symbols are implemented as a network of units: one symbol unit, and n unification units, where n is the number of positions in the network. The symbol unit has strong, bi-directional connections to all the unification units, except the one corresponding to the position the symbol is contained in. Figure 4 shows the structure of one symbol, assumed to be in the first position.

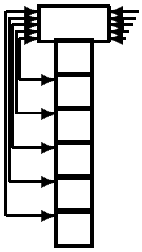

Figure 4: Components of a symbol and their connections

The symbols have to be grouped into two subsets: variables and constant symbols, which include the function symbols. This is necessary because variables play a different role in the unification process, and must have different connections; these are described in the next paragraph.

3.5 Positions

A position is composed of two groups of symbols: variables and constants. Within the group of variable symbols, weak bi-directional connections exist between corresponding unification units of each variable symbol[4]. In addition, each variable symbol has its unification units weakly, uni-directionally connected to the unification units of all constants. The resulting interconnection scheme for the unification units within a position is depicted in Figure 5.

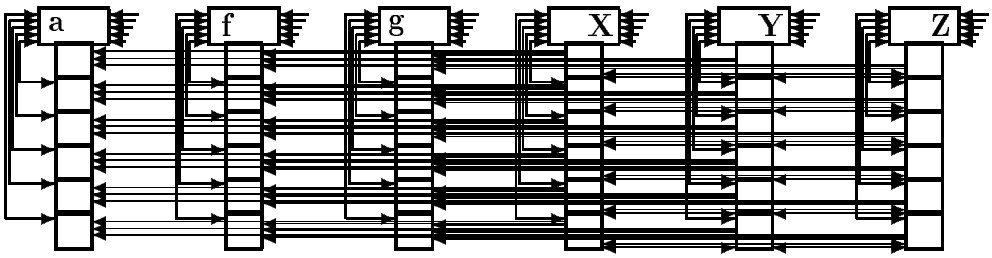

Figure 5: Connections in a position

[4]in the terminology of ICSIM, the unification units are *bus-connected*, and the variable symbols are *completely connected*

3.6 Net

The overall network for unification is composed of positions, their number determined by the structure of the terms to be unified. There are two groups of connections between positions:

1. singularity connections, and
2. decomposition connections.

Singularity connections are needed to establish shared variables; since variables may be shared between any two (or more) positions of the network, each position must be connected to each other. The connections are established for each symbol of a position between a particular symbol unit of position i and the corresponding unification i of all other positions $j \neq i$. These strong, bi-directional connections are explicitly represented for the last symbol, z, in Figure 6, and indicated by arrows for the rest[5].

Decomposition connections represent the term – subterm relations in the unification problem. They are extracted from the particular problem under consideration before network construction, and only the ones really needed are installed. Decomposition involves two father – son pairs which have to be combined (their variable instantiations must be the same) due to a shared symbol in the two father positions, or in predecessors thereof. In our example, the two father positions **0.1** and **0.3** share actually two variables, x and y; this requires identical instantiations for their son positions **0.1.1** and **0.3.1**. Thus the fact that two variables in the father positions are activated together must be propagated to the son positions, and enforce identical activations in the son positions. The common activation of variables in the father positions is mirrored in the activation of the corresponding unification units, in our example unification unit 5 of position **0.1** (which combines it with position **0.3**), and vice versa unification unit 2 of position **0.3**[6]. The propagation of activation to the son positions is achieved through weak, uni-directional connections from these unification units of each symbol at the father position to the respective unification units of all symbols at the son positions; these are unification unit 6 for position **0.1.1**, and unification unit 3 for position **0.3.1**. Together with the parallel connections between all unification units within a position, these decomposition connections achieve identical activation patterns for successors of positions with shared variables; in our example, z is activated at position **0.1.1** and a at **0.3.1** so that both positions have $\{a, z\}$ as active symbols. Figure 7 shows the relevant decomposition connections for our example.

[5] The interconnection scheme as described here actually is somewhat redundant, but was more straightforward to implement.

[6] again, conceptually one of the two unification units is redundant, but used because of an easier implementation

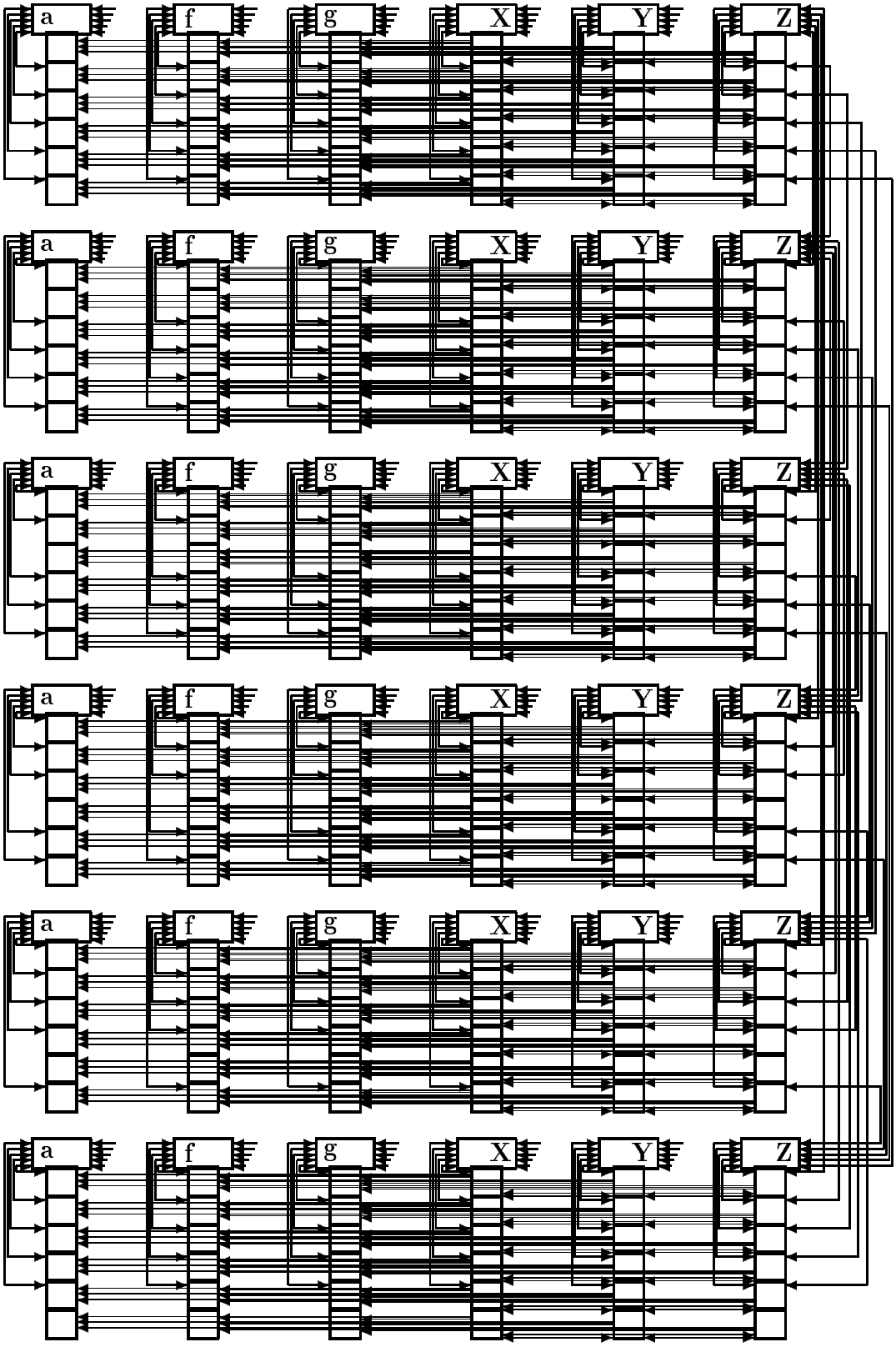

Figure 6: Unification network with singularity connections

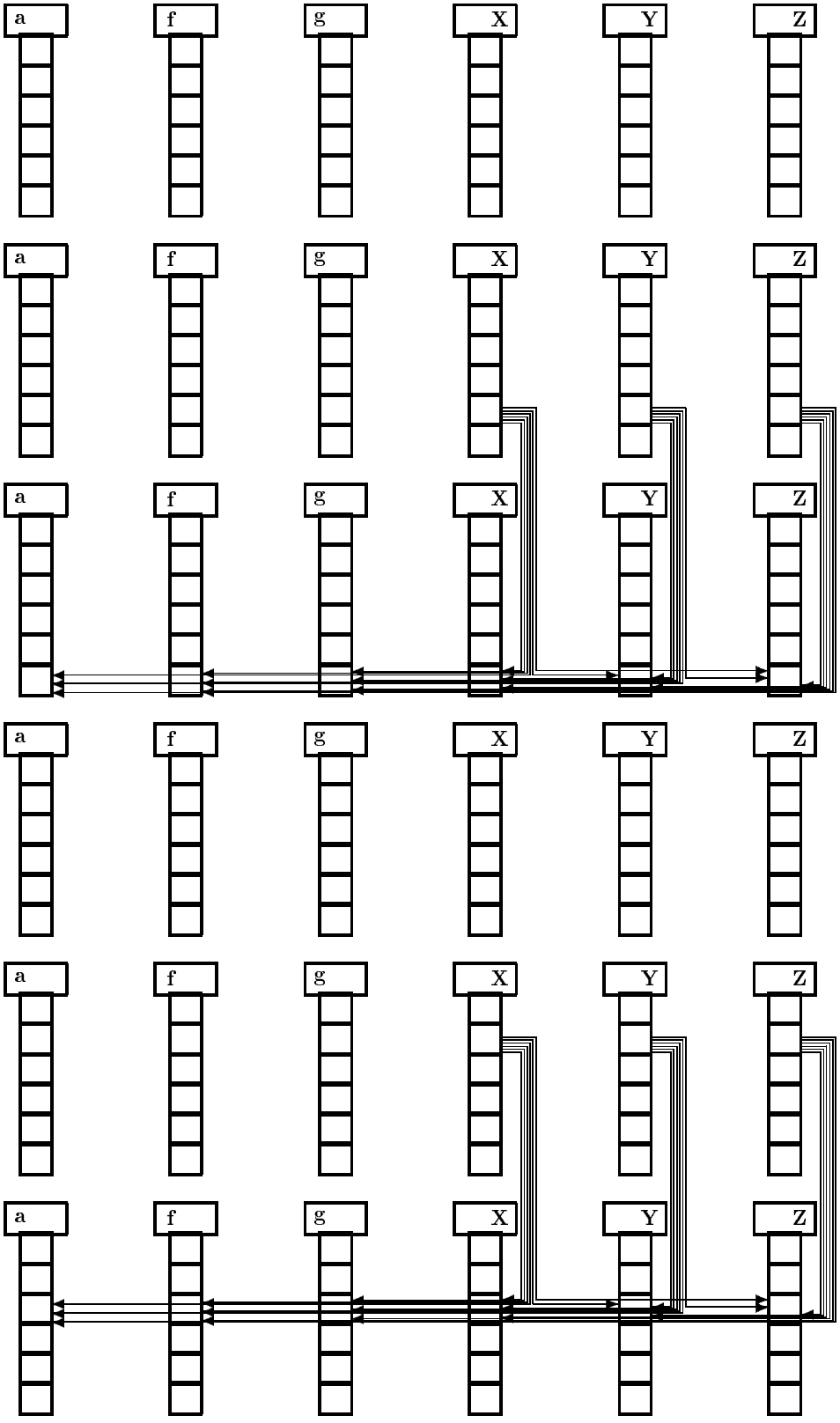

Figure 7: Unification network with decomposition connections

3.6.1 Network Size

The size of the unification network depends on the structure of the terms to be unified; the most relevant parameters are the number of different symbols appearing in the pair of terms, m, and the number of positions in the pair of terms, n. In our example, m and n are both 6.

Number of Units The number of symbol units is the product of the number of symbols and the number of positions: $su = m \times n$. The number of unification units is the product of the number of symbol units times the number of unification units per symbol (which is equal to the number of positions): $uu = su \times n$, or $uu = m \times n \times n$. Thus the overall number of units is $su + uu$, or $m \times n + m \times n \times n$, or $m \times n \times (1 + n)$. The number of units for our example is $6 \times 6 + 6 \times 6 \times 6$, or 252. Note that this is the number of units for the present implementation; it can be reduced by a factor of two without major problems: $m \times n + \frac{1}{2} \times m \times n \times (n-1)$, or $\frac{1}{2} \times m \times n \times (n+1)$ (or 126 in our example) by eliminating redundant unification units.

Number of Connections The determination of the number of connections is a little bit more complicated. For each symbol unit, we have $n-1$ bidirectional connections to its unification units, and also $n-1$ bidirectional connections to one unification unit of the same symbol at the other positions. Bidirectional connections actually are implemented as two one-directional connections, so that the overall number of connections per symbol unit in a position is $c_{su} = 2 \times (n-1) + (n-1)$, or $3n-3$.

The number of connections between unification units in one position depends not only on the number of symbols, but also on the distinction between variable and constant symbols. Let m_v and m_c be the number of variables and constants, respectively. For each unification unit of a symbol, we have connections from each variable symbol to all other symbols: $c_{uu_v} = 2 \times \frac{1}{2} \times m_v \times (m_v - 1)$ is the number of all connections between one unification unit of each variable, and $c_{uu_c} = m_c \times m_v$ for the constants.

The number of connections within a position then is the number of symbols times the connections between the symbol unit and unification unit of a symbol plus half of the connections from the symbol unit to unification units of other positions, plus the number of connections between unification units of the different symbols times the number of unification units per symbol: $c_{pos} = m \times c_{su} + n \times (c_{uu_v} + c_{uu_c})$
$= m \times (2(n-1) + n - 1)n \times (m_v \times (m_v - 1) + m_c \times m_v)$.

The overall number of connections c_{net} in the network is the product of the number of positions times the number of connections in a position, plus some connections between positions which are required by decomposition. Decomposition connections are established between father – son pairs of positions, but are not necessary for the root position and its sons since there is no interesting case where singularity occurs at the root. The positions involved in decomposition hence are only second- or more-

generation successor of the root; their number n_d is the number of all positions n, minus 1 for the root, minus the number of sons of the root n_{sr}. The decomposition connections for one pair of positions go from one unification unit of each variable symbol of the father position to one unification unit of all symbols of the son position; their overall number in the network is $c_d = n_d \times m_v \times m$. Now the total number of connections is given by $c_{net} = n \times c_{pos} + c_d$, which is of the order of magnitude $\mathcal{O}(m^2 \times n^2)$.

For our standard example $< f(x,x,x) = f(g(a),y,g(z)) >$ we have the following figures: $n = m = 6; m_v = m_c = 3; n_d = 2;$
$c_{su} = 2 \times 5 + 5 = 15,$
$c_{uu_v} = 3 \times 2 = 6$, $c_{uu_c} = 3 \times 3 = 9$, $c_{pos} = 6 \times 15 + 6 \times (6 + 9) = 180,$
$c_d = 2 \times 3 \times 6 = 36$, and
$c_{net} = 6 \times 180 + 36 = 1116.$

Please note that these calculations are based on the simple implementation; the number of connections can be reduced by a factor of about two by eliminating redundant unification units.

4 The Execution of the Unification Algorithm in the Network

This section describes the computation of the unification problem and the actions taking place in the network to do so. The discussion is based on the output displayed for the example $< f(x,x,x) = f(g(a),y,g(z)) >$, see Figure 8.

```
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 8: Output for the example $< f(x,x,x) = f(g(a),y,g(z)) >$

The system needs five steps to determine the result, where it reaches a fixpoint. In each step, the relevant units of the network are displayed. Each row represents

units of one position in the network, and the columns are organized according to the symbols occuring in the unification problem, grouped into constants on the left and variables on the right. Each symbol consists of one symbol unit and six unification units. The passive state of a symbol is represented as "**=**", the active state as "**#**"; for unification units, passive is shown as "**-**", active as "**|**". Since the net consists of boolean units, for most cases this representation is sufficient to show the state of the units and the progress of action in the network. For deeper insight, a verbose option is available displaying the activation of a unit as given by the weighted sum of inputs; this is useful for debugging purposes, but it is much more difficult to interpret than the visualization as in Figure 8.

4.1 Initial State

Step 1 shows the initial state of the network: the active units are the symbol units at the positions indicated by the structure of the terms, f at position **0**, g, x at position **0.1**, a at position **0.1.1**, x, y at position **0.2**, g, x at position **0.3**, and z at position **0.3.1**. This is just a straightforward encoding of the information given in the unification problem.

In the second step, the initially active symbol units entail the activation of a number of unification units belonging to the same symbol at the same position. This is due to two strong inputs, one from the symbol occurrence at the same row, another from an occurrence of the same symbol in a different row. In our example, unification unit 5 of symbol g at position **0.1** receives strong inputs from its own symbol unit and from the symbol unit of g at position **0.3**. Unification unit 2 of g at position **0.3** receives strong inputs from the same two occurrences of the symbol, and one of the two unification units actually is redundant. This redundancy appears throughout the net; it causes no harm, and makes the construction of the network easier. In the same way unification units of symbol x are activated.

In step 3, more unification units for symbols a, g, y, z become active, but through different activation paths. The additional flow of activation is caused by the weak bus-connections between the unification units of different symbols in one position: in one row all unification units 1 are weakly connected; bi-directionally between variables, uni-directionally from variables to constants. Consider symbol g, position **0.2**: unification unit 2 receives a strong input from symbol unit g at position **0.1**, plus a weak input from unification unit 2 of symbol x at position **0.2**. These two activations together are sufficient to activate the unification unit. The same combination of intra-symbol (strong input) and intra-position (weak input) accounts for the other activated unification units of symbols g, y.

There is also an activation of unification units at symbols a, z which cannot be explained by the mechanism above. Here we encounter an example of decomposition, the father - son relation between positions. Let's consider the activation of unification unit 3, symbol a, position **0.3**: it recieves strong input from the active symbol unit of a at position **0.1** (row 3); the additional weak input comes from any active unification

unit 2 (here at g or x) at its father position. The active state of unification unit 2 expresses the fact that the corresponding symbol unit (at position 2) is active, and both occurrences of this symbol must have the same value. This must be propagated to the son positions, here positions **0.1.1** and **0.3.1**, and thus activates unification unit 3 of a at position **0.3.1** and unification unit 6 of z at position **0.1.1**.

Step 4 shows the activation of some more unification units, and the activation of five symbol units at symbols a, g, y, z. At this point all symbol units have been activated; step 5 only shows the activation of some more unification units without any further effects.

4.2 Singularity

The effect of singularity can in addition be visualized by overlaying a rectangle on the network: its corners are active symbol units of a pair of symbols at a pair of positions, e.g. g, x at positions **0.2, 0.3**. During the evaluation of the network, "missing" corners of incomplete rectangles are established whenever two symbols must have the same value at two positions.

4.3 Decomposition

Decomposition can be visualized by a parallelogram, the corners being active symbol units of two symbols across pairs of father - son positions, e.g. x at the father positions **0.1, 0.3** and z at the son positions **0.1.1, 0.3.1**. A missing corner is established here due to the propagation of the unification task from father to son positions.

4.4 Determination of the Result

In most cases, unification will not be used as a high-level operation with a direct interface to the user. It is typically a low-level operation embedded in a more complex system, e.g. for drawing logical inferences. In such a context, it is important to represent the result of the operation in a way which is consistent with its further usage. In our case, the essential information is expressed by the values the variables in the unification problem assume after its execution. A typical representation of these values is in the form of a *most general unifier* for the terms involved. This information can be derived from the active symbol units in the final state of the network.

The main idea is to identify the equivalence[7] classes of the positions occuring in the problem, and then to construct the variable substitutions according to the variables and constants whose symbol units are active in a particular position.

[7]The notion of equivalence class actually is a little sloppy here. To be precise, the underlying equivalence relation between positions is a decomposable, singular equivalence relation, or *DSE-relation*; this states that corresponding subterms are equal, and that shared variables are equal. Furthermore, we assume (or check it explicitly) that our unification problem is homogeneous, i.e. each position contains at most one active constant symbol.

The determination of equivalence classes from the final state of the network is quite straightforward: two positions are in the same equivalence class iff they have the same set of active symbols. For our standard example, the equivalence classes together with the set of active symbols as labels are $[0]\{f\}; [0.1, 0.2, 0.3]\{g, x, y\}; [0.1.1, 0.3.1]\{a, z\}$.

Deriving the substitutions from the final state of the network is a little more complicated, mainly due to the fact that variables can be assigned the value of an expression corresponding to a subterm. As a consequence, parts of the original term structures may have to be reconstructed from the network. If an equivalence class is labeled by variables only, one of them is selected to form a substitution, e.g. $\sigma = \{x \leftarrow y\}$, or $\sigma = \{x_1 \leftarrow x_2 \leftarrow x_3 \ldots x_n\}$. Otherwise, the label of a class contains exactly one function symbol (or constant), and the substitution consists of the constant representing the function symbol, plus expressions derived from the arguments of the function, e.g. $\sigma = \{x \leftarrow g(arg_1, arg_2, \ldots, arg_m\}$; the determination of the substitutions must be continued for the arguments. If a class is labeled only by a function symbol, and no variables, it does not affect the substitutions. After all equivalence classes have been checked, the most general unifier is given by a set of substitutions, e.g. $\sigma = \{z \leftarrow a, x \leftarrow y \leftarrow g(a)\}$.

Informally, the network is compressed by merging rows and columns with identical patterns of active symbol units (unification units are irrelevant here). In our example, rows 2, 4, 5 can be merged as well as rows 3 and 6; the columns which can be merged are a, z and g, x, y.

5 Future Work

The present implementation of the unification algorithm suffers from a number of deficiencies, some due to the simplistic implementation, others due to the underlying algorithm, and again others due to the approach to unification.

Implementation The most obvious improvement is to get rid of the redundant unification units combining symbols in different positions which might become unified. The implementation as described here uses n^2 unification units; this can be reduced to $\frac{1}{2} \times n \times (n-1)$ by moving the unification units away from the corresponding term units and grouping all unification units connecting the different occurrences of a symbol together. As a consequence, the number of connections would also be reduced substantially.

Algorithm The number of units required for the algorithm as described in [Hölldobler, 1990b] still is quadratic: $\frac{1}{2} \times m \times n \times (n-1)$. Many of these units, however, will never be activated during the execution of the algorithm. This can be used for an incremental construction of the unification network. Assume that we have a pool of unification units and term units, completely connected with very low weights. As the unification problem is parsed, the term and unification units are allocated and the weight of the connections is increased to the value corresponding to weak and strong connections described previously. After the construction of the network, additional term and unification units are allocated as required through the combination of already active units. In the worst case, if all occurrences of all symbols have to be unified with each other, this would still require the same number of units as above; for realistic cases,a substantial reduction can be expected. For the (relatively small) examples studied, roughly $\frac{1}{2}$ to $\frac{1}{5}$ of the term units were actually used, and about $\frac{1}{5}$ to $\frac{1}{10}$ of the unification units.

Different Approaches There are a number of different approaches to tackle the unification problem with connectionist means. One is to view the unification problem as a graph, e.g. as a dag (directed acyclic graph) and perform certain operations on this graph. One problem here is that there are two relations in the unification problem which must be represented in the graph: decomposition and singularity. This can be done by introducing two different types of links between the nodes.

Another possibility is to use a graph consisting of nodes for symbols, positions and arguments; the fact that a symbol appears at a certain position is expressed by an arc (solid lines) between a symbol node and the corresponding position node, and the function - argument relation is expressed by arcs (dotted lines) between two positions. Additional arcs between symbols and between positions are introduced during evaluation, indicating that two (or more) symbols must have the same value, and two (or more) positions belong to the same equivalence class. Two problems

in this case are the multiple occurrence of function symbols at different positions, which may have different evaluation results, and the propagation of unification from terms to their arguments. Both problems might be resolved by the introduction of additional nodes, e.g. one node per occurrence of a function symbol. The evaluation of such a network may either proceed stepwise per equivalence class, or use different output values for different equivalence classes, or separate units fo equivalence classes.

Our standard example $< f(x,x,x) = f(g(a),y,g(z) >$ then looks roughly as depicted in Figure 9.

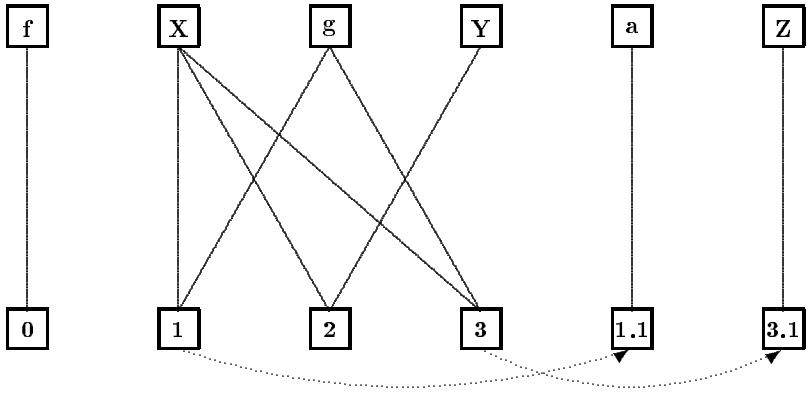

Figure 9: Unification via equivalence classes: Initial state

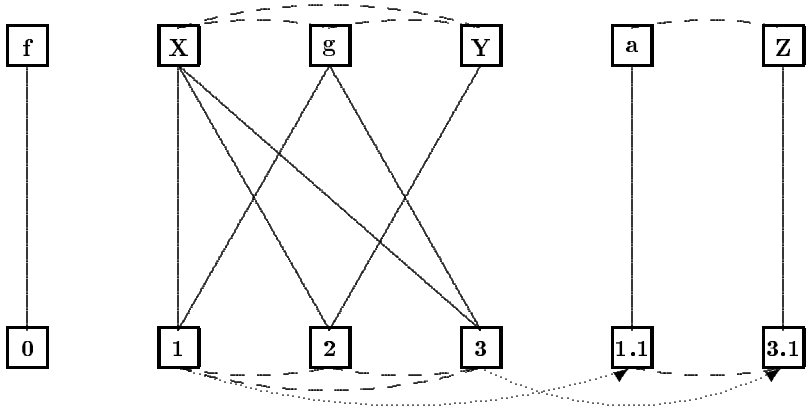

Figure 10: Unification via equivalence classes: Final State

After the evaluation, there are additional links (depicted as dashed lines) within the symbol and position layers (see Figure 10). These links connect symbols and positions which are in the same equivalence class. If two positions π_1 and π_2 are in the one equivalence class, then their corresponding children $\pi_1.\pi$ and $\pi_2.\pi$ (the

arguments of the functions) must also be in one equivalence class. An idea somewhat related to this graphical approach is based on a dag representation [Bibel et al., 1987]. It relies on two operations: one is to "melt" nodes with the same variable symbol at different positions (singularity), and the other to "forward" the task of unification according to the term - argument relation (decomposition). Problems here are to "melt" nodes, and to differentiate between the different arguments of a function. The melting of a set of nodes can be achieved either by coloring different sets with different colors, or by isolating all the nodes of a set but one, and establishing new connections for this node in place of the isolated ones.

A third approach is to view the unification problem as a sequence of symbols, and use a recurrent network to analyse if the sequence is admissible or not [Elman, 1989, Elman, 1990]. A problem is the determination of the variable substitutions, which certainly are of interest. Viewing unification not as an operation on a sequence of symbols in the order they are usually written down, but as a sequence of pairs of functions might be a way to use recurrent networks; however, this does not seem to help a lot for the determination of variable substitutions. First experiments in

Example: $< f(x,x,x) = f(g(a),y,g(z) >$ is presented as

```
      f         f
  x x x     g y g
            a   z
```

Figure 11: Unification as sequence of function pairs

this direction have not been very successful; further evaluation is needed, however, to determine the reasons for the problems: Are they of a fundamental nature, i.e. is unification not feasible with this type of network, or is the setup of the experiments inadequate. One possibility, for example, is that the database of unification examples used is too small.

A similar approach with more promising results (for a simplified version of unification, however), has been investigated by [Stolcke and Wu, 1991]. The basic idea there is to employ a recursive auto-associative memory [Pollack, 1988, Pollack, 1990] for the representation of the term structure, and modify it in such a way that it also performs unification.

6 Conclusions

The major goal of this endeavor in unification with connectionist techniques has been to demonstrate that the algorithm works. Although it is far from being competitive with conventional unification algorithms based on assignment of values to variables and manipulation of pointers, its advantages lie in an easy integration into systems with other connectionist modules, e.g. front-ends from image or speech processing.

ICSIM as a development tool is suited for this kind of work. It does involve programming, however, and in particular the relatively complicated interconnection patterns for the unification network are somewhat tedious to establish (which is not a problem of ICSIM but a feature of the unification mechanism). Currently ICSIM is being enhanced in tow directions. One is the coding of further basic components and elementary networks for common types of connectionist architectures, the other a more sophisticated user interface with graphics output. Especially the latter feature is very desirable for the unification network, in order to dynamically display the activities going on during execution.

The unification network described here actually is the core of a connectionist inference mechanism [Hölldobler, 1990a]. This inference mechanism relies on the same basic representation and processing techniques, and an implementation of it, also using ICSIM, is under way.

References

[Bibel et al., 1987] Bibel, W., Kurfeß, F., Aspetsberger, K., Hintenaus, P., and Schumann, J. (1987). Parallel inference machines. In Treleaven, P. and Vanneschi, M., editors, *Future Parallel Computers*, number 272 in Lecture Notes in Computer Science, pages 185–226, Berlin. Springer.

[Elman, 1989] Elman, J. L. (1989). Structured representations and connectionist models. In *COGSCI '89*, pages 17–23.

[Elman, 1990] Elman, J. L. (1990). Finding structure in time. *Cognitive Science*, 14:179–211.

[Hölldobler, 1990a] Hölldobler, S. (1990a). CHCL – a connectionist inference system for horn logic based on the connection method. Technical Report TR-90-042, International Computer Science Institute, Berkeley, CA 94704.

[Hölldobler, 1990b] Hölldobler, S. (1990b). A structured connectionist unification algorithm. In *AAAI '90*, pages 587–593. A long version appeared as Technical Report TR-90-012, International Computer Science Institute, Berkeley, CA.

[Meyer, 1988] Meyer, B. (1988). *Object-Oriented Software Construction*. Prentice Hall, New York.

[Omohundro, 1990] Omohundro, S. (1990). The Sather Language. Technical report, International Computer Science Institute.

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

[Pollack, 1988] Pollack, J. (1988). Recursive auto-associative memory: Devising compositional distributed representations. In *10th COGSCI*.

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

[Schmidt, 1990] Schmidt, H. W. (1990). ICSIM: Initial design of an object-oriented net simulator. Technical Report TR-90-55, International Computer Science Institute, Berkeley, CA 94704.

[Stolcke and Wu, 1991] Stolcke, A. and Wu, D. (1991). Tree matching with recursive distributed representations. International Computer Science Institute.