Quantification without Variables in Connectionism[*]

John A. Barnden

Computing Research Laboratory & Dept. of Computer Science
Box 30001, Dept. 3CRL
New Mexico State University
Las Cruces, NM 88003-0001
(505) 646-6235 jbarnden@nmsu.edu FAX: (505) 646-6218

&

Kankanahalli Srinivas

Concurrent Engineering Research Center &
Department of Computer Science
West Virginia University
Morgantown, WV 26506
(304) 293-7226 ext 137 srini@cs.wvu.edu FAX: (304) 293-7541

May 8, 2015

SEND CORRESPONDENCE TO: **Srinivas.**

SUBMITTED TO *Mind and Machines*

KEYWORDS: variables, quantification, systematicity, combinators, reduced representations, symbolic representations, working memory, inference

[*]This work was supported in part by grant AFOSR-88-0215 from the Air Force Office of Scientific Research, to Barnden, and grant NAGW-1592 under the Innovative Research Program of the NASA Office of Space Science and Applications, to Barnden and C.A. Fields.

Abstract

Connectionist attention to variables has been too restricted in two ways. First, it has not exploited certain ways of doing without variables in the *symbolic* arena. One variable-avoidance method, that of logical combinators, is particularly well established there. Secondly, the attention has been largely restricted to variables in long-term rules embodied in connection weight patterns. However, *short-lived* bodies of information, such as sentence interpretations or inference products, may involve quantification. Therefore, short-lived activation patterns may need to achieve the effect of variables. The paper is mainly a theoretical analysis of some benefits and drawbacks of using logical combinators to avoid variables in short-lived connectionist encodings without loss of expressive power. The paper also includes a brief survey of some possible methods for avoiding variables other than by using combinators.

1 Introduction

Connectionist attention to variables has been too limited in two senses: (1) it has not considered exploiting ways that the *symbolic* arena itself provides for doing without variables; (2) connectionist attention has largely been restricted to variables in long-term rules (embodied in connection weight patterns), even though *short-lived* bodies of information (embodied in short-lived activation patterns) may need to achieve the effect of variables. Examples of such short-lived items are sentence interpretations and inference products.

The present paper seeks to go beyond limitations (1) and (2). It mainly presents a theoretical analysis of some benefits and drawbacks of trying to avoid variables in short-lived connectionist encodings by using one particular variable-avoidance method that is particularly well established in the symbolic arena. The method is that of *logical combinators* (Schönfinkel 1924, Curry & Feys 1958). We stress that we are not claiming that the combinator method should definitely be adopted.

Consider the following rule:

> IF *some person X loves another person Y, and Y loves a third person Z,*
>
> THEN *X is jealous of Z.*

A cognitive system, connectionist or otherwise, might have this as part of its *long-term* knowledge, in the sense either of following it or at least acting (approximately) in accordance with it. The connectionist concern with variables has largely been fixated on the variables, like X, Y and Z, in such rules. The technical problem then becomes either (i) that of devising ways of implementing rules and variables in some direct way in a connectionist system, or, more indirectly, (ii) that of devising connectionist processing mechanisms that behave as if they have variabilized rules, but don't actually have any nodes, links, assemblies, activation patterns, etc. identifiable as variables.

The main point, clearly, is that we want to avoid having a separate connectionist subnetwork for each possible combination of values for the variables. Ideally, we want a single subnetwork that deals with all cases, although we might tolerate some limited number of networks, each dealing with some broad subclass of cases. That is, we want to achieve *generality* of processing in a *compact* way. Notice that this desire is perfectly consistent with not wanting any direct implementation of rules and explicit rule following in a connectionist system. Any (non-toy) connectionist system, of whatever style, that is meant to conclude that a person is jealous of another person when they are involved in a love triangle as above must exhibit compact generality in its inferential mechanisms. (From such considerations, we see that the issue of compact generality overlaps the notion of systematicity of inference, as described in Fodor & Pylyshyn, 1988.)

Even if explicit rule following is directly implemented in a connectionist system, it does not necessarily follow that *variables* must be implemented in order to achieve compact generality in the rules. One reason for this is that there are methods for achieving compact generality without having variables, and there is at least one such method (the combinator approach) that preserves the full expressive/inferential power that variables provide, without even departing from the symbolic framework. This is not to say that variables are not ultimately the method of choice — we point out variable-free methods partly just to expand the theoretical limits of connectionist research on reasoning.

With respect to limitation (2) mentioned above, Hadley (1993a) and Barnden (1992) have noted that connectionist research on rules has concentrated on rules as *long-lived* items (items in the listener's long-term knowledge). However, they can be short-lived items instead. They can, for instance, be communicated in natural language sentences, as when someone says that *if you meet an avid cricket fan you shouldn't mention baseball.* Clearly, the sentence exhibits compact generality, since it applies to any cricket fan. Now, although it is possible that the information conveyed by this sentence ends up in the listener's long-term memory, the listener may *immediately* need to use the uttered rule in an inference – that is, before it the rule can plausibly have entered long-term memory. The speaker's next sentence might be "Stephen here is a great cricket fan," in which case the listener should infer *now* that it should keep quiet about baseball. Thus, the listener needs to construct a short-lived representation of the sentence's meaning, for immediate use, as well as (or instead of) a long-term knowledge item. It is clear that the creation of a long-term item is not always to be expected: for one thing, the listener may rapidly forget what the speaker said; for another, the listener may disagree with the speaker's rule and therefore not include the rule in its own rule base.

Also, compact generality might be needed in: (a) short-lived representations inferred from other short-lived representations; (b) short-lived representations derived from input forms other than natural language — for example, if the system, in a zoo, sees some monkeys eating bananas, it might construct a short-lived proposition to the effect that "every monkey here is eating a banana". This proposition contains existential compact generality embedded within universal compact generality.

These points about short-lived representations are obvious from the point of view of symbolic artificial intelligence, but for historical reasons they have not had much effect on connectionism other than in connectionist work on theorem proving and unification (see, e.g., Hölldobler 1990, Pinkas 1992). Even systems aimed at implementing symbol manipulation (i.e., "implementational" connectionist systems, in the terminology of Pinker & Prince 1988) have largely dealt with short-lived representations that do not have features of compact generality. Harris and Elman (1989) point out the importance of the distinction between variable-like elements in sentences and variables in inference mechanisms that operate on sentences; to our knowledge, however, significant quantification-sensitive inference has not been addressed by research related to the Harris & Elman approach. (We should also note that the role-binding problem has often been amalgamated with the variable binding problem, and is in part concerned with role-binding in short-lived representations. However, in this paper role binding only plays a subsidiary role, because in itself it does not generally involve compact generality.) Therefore, there is a real question as to whether current connectionist techniques will readily be able to deal with compact generality in short-lived encodings.

Since the issue of compact generality is not confined to the question of rules or other inferential mechanisms, it is broader than Fodor and Pylyshyn's notion of systematicity of inference. Notice also that the examples of compact generality given above all exhibit a "universal" variety of compact generality: e.g. the jealousy rule above could be seen as having the force of the universally quantified proposition *for all* X, Y and X, IF ... THEN . However, short-lived propositions or long-term knowledge items might involve existential compact generality — corresponding to existential quantification — instead of, or as well as, the universal variety. Note we prefer to avoid the term "quantification" in favor of "compact generality" for two reasons: (i) someone may view quantification as presupposing variables, whereas we wish to avoid assuming that variables are needed in order to make general statements; and (ii) the jealousy rule as originally stated does not overtly contain quantification.

Throughout the paper we will make the simplifying but common assumption that a connectionist system holds short-lived information in its temporary activation patterns, rather than in, say, rapidly-changed weights. The main problem therefore is to get *connectionist activation patterns* to exhibit the degree of compact generality that is patently necessary for mundanely realistic high-level cognition, such as natural language understanding or common-sense reasoning. Now, within the traditional symbolic framework there is not much point in distinguishing between the short-lived and long-lived versions of the compact generality issue. Short-lived and long-lived symbolic representations have much the same sort of implementation (as linked lists or whatever), even when the long-lived representations are procedures rather than declarative facts. Therefore, if a given device for compact generality, such as variable-based quantification, is used in long-lived representations it can readily be transferred to short-lived ones, and vice versa. This contrasts crucially with the case of connectionist systems, because long-lived items are (usually) encoded in the weights whereas short-lived items are (usually) encoded as activation patterns. Thus, connectionist methods that have been developed for achieving compact generality in long-lived rules may provide little guidance for the case of short-lived items, or vice versa.

A connectionist system that did hold short-lived information in short-lived alterations of its weights would be less susceptible to the divide we have noted between short-term and long-term representation. Of course, it would have to be shown that such a system could indeed rapidly manipulate its short-lived representations in desired ways, and achieve the right degree of compact generality. Such a system could perhaps be made to work. However, our motive in the previous paragraph is simply to point out that, to the extent that the connectionist field does tends to think of short-lived information as residing only in activation patterns, it exacerbates the particular problem of dealing with compact generality in short-lived representations.

The plan of the paper is as follows. Section 2 will briefly survey some conceivable methods for achieving compact generality in short-lived representations. None of these is as well developed theoretically and computationally as the combinator method, but they all present interesting possibilities. Section 3 will sketch the combinator method for achieving compact generality, concentrating on the case of short-lived encodings. The section will briefly mention some advantages and disadvantages in comparison to variables, from the connectionist point of view. Section 4 presents some additional discussion, and section 5 is the conclusion.

Our aim is not to argue that connectionists should definitely use combinators. Therefore, we concentrate on theoretical considerations rather than on providing an implementation of a connectionist combinator-based system. Still, we will comment briefly on various concrete ways in which combinators might be imported into connectionism, and briefly mention some preliminary implementational work we have done. Our hope is that the discussion of combinators will will help to guide future work on connectionist systems that achieve compact generality, whether by means of variables, combinators, or some other device. Combinators are the most fully developed way of achieving compact generality without variables. Therefore they constitute the method about which the most definite and precise statements can currently be made. Combinators have long been used within computer science and symbolic logic to avoid variables, and they have been fundamental in part of the programming language subarea of computer science (Turner 1979, Peyton Jones 1987).

2 Some Variable-Free Methods

Here we briefly look at some ways in which compact generality in short-lived representations could perhaps be achieved without using variables. They are less fully developed than the combinator method to be described in section 3, in the sense that they are preliminary suggestions or are limited in their scope.

2.1 A Possible Technique Derived from Natural Language

Compact generality in natural language does not ordinarily use anything like overt variables (although of course we can use dummy names like "X" in sentences if we wish). For instance, there is no overt variable-like entity in the sentence *"There is a bank in Georgia that gives each of its customers a $10 pizza coupon every year."* So, perhaps a good strategy is to import natural languages' compact generality techniques into connectionism. Some existing non-implementational connectionist work might lead the way towards this. The work is typified by that in Harris & Elman (1989); see also St. John & McClelland (1990) for closely related research. The systems learn their own internal encodings of natural language sentences that appear on an input layer. The learned hidden-layer encodings are a form of "compressed encoding" (or "reduced representation" or "reduced description" — Hinton 1990).[1] They apparently cannot at all easily be viewed as implementations of formal-symbolic, internal representations of sentences. Now, since no intervening level of symbolic structure is designed into the system, we might wonder whether the learned

[1]We will use the term "compressed encoding" although it is less common. We wish to avoid confusion with the notion of "reduction" in combinator theory, to be discussed below.

encodings in such systems could (some day) contain adequate compact generality without containing any feature that could readily be viewed as a variable. However, although such systems have demonstrated some interesting capabilities, including ones that might be called inferential, they are very far from being established as adequate for inference involving compactly general premises and conclusions.

Also, Hadley (1994) notes that the target items used in the training in St. John's system have formal compositional structure, and it is quite possible that our imagined extension to handle compact generality would require those structures to become committed to compact generality devices taken from a non-natural-language symbolic framework. This would defeat the main point of our suggestion if it meant that the system's compressed encodings turned out to approximate those devices.

2.2 Borrowing Natural Language Techniques

There is another conceivable way of borrowing tools for compact generality from natural language, although now without necessarily encoding natural language fragments as such. (It is somewhat reminiscent of the indexical-functional scheme in Chapman & Agre 1987.) Part of the point of variables in formal logic is that they provide an easy way of referring to the same indefinite entity more than once in the same expression. The same purpose is achieved in English by various types of definite description, including some we will call *meta-linguistic.* Consider the following sentence: *"If some person loves another person, the second one loves a third, and all three people are different, then the first is jealous of the third."* The phrases "the first," "the second one" and "the third" refer back *meta-linguistically* in the English sentence. We say meta-linguistically, because the phrases appeal to particular *mentions* of people. In the example, the phrase "the third person" does not denote the third person in some existing sequence of people in the world; it is more as if it were an abbreviation for "the person I meant the third time I mentioned a person." Indeed, the phrases "the first person" and "the third person" could, in a different example, refer to the same individual.

Now, one could imagine a formal symbolic structure which closely parallels the form of the English sentence above. It could be something on the lines of the following:

```
loves(any(person), any(person)) ∧ loves(2nd(person), any(person))
  ∧ different(1st(person),2nd(person),3rd(person))
```

$\rightarrow$ `jealous(1st(person),3rd(person)).`

The symbols `1st`, `2nd` and `3rd` are special operators for building terms that refer back *meta-logically* on the basis of the structure of the very formula they appear in. A connectionist system could manipulate activation patterns that exactly or approximately encoded formulae like the ones imagined. They would thereby avoid anything like conventional logical variables and quantifiers, while still providing something like the same power.

2.3 Johnson-Laird Mental Models

We could possibly avoid variables in short-lived representations by using some extended version of Johnson-Laird's mental model theory (Johnson-Laird, 1983; Johnson-Laird & Byrne, 1991). In this theory, compactly general natural language statements ultimately get translated into "models" in which quantification over a class of individuals is replaced by the use of a set of tokens that stand for arbitrary, imaginary individuals in the class. For instance, consider the proposition that *some of the athletes are beekeepers.* A possible mental model for this is as follows, using `A` and `B` to stand for the class of athletes and the class of beekeepers respectively:

A:x_1	B:x_1
A:x_2	B:x_2
(A:x_3)	
	(B:x_4)

A mental model is an unordered set of *tokens*, shown above as the T:x_i items. Each x_i is an arbitrary label for a member of that class. If two tokens contain the same label x_i, then they stand for the same entity. Otherwise, they stand for different entities. As the proposition says that some of the athletes are beekeepers, some non-empty subset of the athlete tokens share labels with some of the beekeeper tokens. Parentheses indicate that the enclosed tokens are "optional." If all tokens are non-optional, then the mental model is a description of a single example situation conforming to the proposition(s) in question. If there are optional tokens, the mental model describes several example situations, differing on which of the optional tokens are deemed actually to denote entities.

Mental models are stated to contain no variables (Johnson-Laird & Byrne, 1991: pp.144, 212). Johnson-Laird and Byrne show how a wide variety of types of inference can be handled by mental model manipulation. Much of this work concerns compactly general, short-lived information. Also, mental model manipulation can be straightforwardly realized in suitably structured connectionist systems, as is shown by the direct connectionist implementation detailed in Barnden (1994, 1995) (see also Barnden 1989).

Let us grant for the sake of argument that some extension of Johnson-Laird's approach can account for all needed types of inference over compactly general, short-lived pieces of information. The theory would not, however, get rid of variables unless natural language inputs (etc.) were converted directly into mental model terms, with no intervening logical form containing variables; but to our knowledge no such direct-conversion mechanism has been proposed. Indeed Johnson-Laird and Byrne (1991) set forth a sentence-to-model conversion scheme that constructs intermediate "semantic representations," and when the input sentence contains compact generality, the semantic representation contains quantifiers and variables (p.177).

2.4 Semantic Networks

Another way we might think we could avoid variables is to use semantic networks. That is, the connectionist system's short-lived representations would be (approximate or exact) encodings of semantic network fragments, rather than of logic expressions. In couching the content of a logic expression as a semantic network fragment, all occurrences of a given variable can be replaced by links to a single node. See, for instance, the node labeled "x" in Figure 1. Let us call such nodes "variable nodes."

FIG. 1 ABOUT HERE

The use of the label "x" in the figure is just for purposes of reference in our discussion — it is not a symbol in the system itself, and is therefore not like a logic variable name. Nevertheless, in graphic depictions of semantic networks, a variable *node*, with its multiple incoming links, is just a diagrammatic analogue of multiple occurrences of a variable name in a logic formula. For instance, what corresponds to variable substitution is now an operation of replacing all those links by links to some other node. We will discuss the semantic network approach further in section 4.

3 Combinators

Here we explain how combinators might be used to avoid variables in short-lived representations in connectionist systems. Sections 3.1 to 3.6 explain a particular version of the combinator approach, but staying within the symbolic framework. Section 3.7 mention some possible ways of transporting the technique into connectionism. Sections 3.8 and 3.9 discuss advantages and disadvantages of doing so.

3.1 Basics

In the rest of the paper we will adopt some notational conventions that are standard in the area of combinatory logic.

We use juxtaposition to stand for function application. Thus, `square` x replaces the more familiar forms `square(`x`)` and x^2. Also, `plus` $e_1 e_2$ for two expressions e_1 and e_2 means the same thing as $e_1 + e_2$. In fact, `plus` $e_1 e_2$ is to be parsed as `(plus` e_1`)`e_2. The idea here is that `plus` is actually a one-place function that, when applied to a number n, delivers a function that adds n to *its* single argument. (This function is called the "curried" version of the more normal two-place addition function.)

Association is to the left in expressions, so that for instance $\mathbf{S}fgh$ is to be interpreted as $((\mathbf{S}f)g)h$. This denotes the result of applying the function **S** to f, thereby getting a function that is applied to g, thereby getting a further function that is applied to h. Thus the same effect is obtained as having a three-place function, taking arguments f, g, h.

Schönfinkel (1924) pointed out the desirability of using combinators to eliminate variables from logic, thereby streamlining the expressions and manipulations. Combinators are functions of a very broad class, but we will rely only on a very specific subset. The single argument to each combinator is often a function, but in some cases it can be an ordinary value. Also, the result of applying a combinator is often another function. In the following definitions of the combinators **I, K** etc. that we will be using, arguments x and y can be any values, whereas arguments f and g can only be functions.

$\mathbf{I}x = x$

$\mathbf{K}xy = x$

$\mathbf{B}fgx$ = $f(gx)$

$\mathbf{B}_n f_1 f_2 \ldots f_n x$ = $f_1(f_2 x)\ldots(f_n x)$ (for $n \geq 2$).

I is the *identity* function. $\mathbf{K}x$ is the *constant* function which always delivers x. Thus, **K** is a function that constructs constant functions. **B** forms the *composition* of two functions. Note that $\mathbf{B}_2$ is just **B**. However, for $n > 2$, $\mathbf{B}_n f_1 f_2 \ldots f_n$ is *not* the composition of the n functions. Rather, it *distributes* an argument x to the $f_2, \ldots, f_n$, and then applies f_1 to those function applications. We will only use go up to $n = 4$ in examples below.[2]

As a simple example giving the flavor of the combinator approach, consider a function `p` that would normally be defined as follows:

`p`x = `plus (log` x`) (square` x`)`

(i.e. $\mathtt{p}(x) = \log x + x^2$). An equivalent variable-free definition is

`p` = $\mathbf{B}_3$ `plus log square.`

To see why this is, consider applying `p` to the argument `13`. We get $\mathbf{B}_3$ `plus log square 13`, which is just `plus (log 13) (square 13)`, which means the same as $\log 13 + 13^2$. In this derivation we "reduced" the $\mathbf{B}_3$ expression, i.e. we applied the definition of $\mathbf{B}_3$.

Our framework for discussion will be a system that manipulates short-lived expressions that can include combinators but no variables. Notice that we can think of the above definitions of the combinators as specifications of long-lived rules for manipulating combinator expressions. These rules are therefore described using variables — f, x, etc. However, our focus of attention in this paper is the avoidance of variables in *short-lived* expressions.

A combinator reduction machine is a symbolic system that works by repeatedly performing the various reductions associated with the combinators. In a sense, the machine may perform the reductions on an given expression in any order, since the same result arises upon termination regardless of the sequence of reductions. However, some sequences may terminate faster, while others may not terminate. To ensure termination, usually the left-most reducible expression is processed first. This is known as normal order reduction (see, e.g., Turner 1979). However, our discussion below is not strongly dependent on any specific reduction order.

[2]The $\mathbf{B}_n$ for $n > 2$ are used in this paper because of our special purposes. They can be expressed in terms of **K**, **I** and another well-known combinator, **S**, but allow simpler expressions.

3.2 Powers of Combinators

We will make much use of powers of combinator expressions, notated by superscripts. The *nth* power of a (function-valued) combinator expression f is f composed with itself $n-1$ times. For instance, $\mathbf{K}^2x$ is the same as $\mathbf{K}(\mathbf{K}x)$. Similarly, $\mathbf{K}^3$ is the composition of $\mathbf{K}$ and $\mathbf{K}^2$ (and of $\mathbf{K}^2$ and $\mathbf{K}$), so that $\mathbf{K}^3x$ is the same as $\mathbf{K}(\mathbf{K}(\mathbf{K}x))$.

Note carefully that $\mathbf{K}^2$ is very different from $\mathbf{KK}$. The latter is the *application* of $\mathbf{K}$ to itself. An expression of form $\mathbf{KK}x$ means $(\mathbf{KK})x$ by left associativity, and this reduces simply to $\mathbf{K}$.

Of course, f^nx can always be written out as nested aplications of f. But it is important to realize that we will often need an expression of form f^n as a complete expression, i.e., not applied to anything. For instance, we will need $\mathbf{K}^2$ as a complete, unapplied expression. Now, since powers of combinators amount to repeated composition, and $\mathbf{B}$ is the composition combinator, it easily follows that for any function-valued combinator expression f, f^n denotes the same function as does $(\mathbf{B}f)^{n-1}f$. Since the latter can be written out as nested applications of $\mathbf{B}f$, without the use of power notation, we see that power notation can be dispensed with entirely. However, for simplicity we will take combinator notation to include powers.

By convention, we take f^0 to denote the same function as $\mathbf{I}$. We assume that a reduction machine does the following:

it rewrites any expression of form f^0 as $\mathbf{I}$;

it rewrites f^1 as f;

for $n > 1$, it can rewrite f^na as $f(f^{n-1}a)$ and vice versa.

3.3 Combinators and Inference

Our concern is not with arithmetical calculation, but rather with such matters as inference over compactly general representations arising from, say, natural language utterances. Consider the sentence *"Short women are humorous."* In a conventional logical style this could be expressed as

$(\forall x)$ `is-short(`x`)` $\wedge$ `is-woman(`x`)` $\Rightarrow$ `is-humorous(`x`)`.

An equivalent combinator expression is:

`ALL(`$\mathbf{B}_3$ `IMPLIES (`$\mathbf{B}_3$ `AND is-short is-woman) is-humorous).`

The function `ALL` takes a predicate p as argument and returns `true` if and only if p always delivers `true` no matter what it is applied to. (A predicate is merely a function that delivers `true` or `false`.) So, `ALL red` returns true if and only if everything in the domain of discourse is red. The function `IMPLIES` corresponds to the ordinary material implication. `IMPLIES`ab, where a and b evaluate to truth values, delivers `false` if a yields *true* and b yields `false`, but otherwise delivers `true`. This can be captured as follows:

`IMPLIES true =` $\mathbf{I}$

`IMPLIES false =` $\mathbf{K}$ `true.`

Similarly,

`AND true =` $\mathbf{I}$

`AND false =` $\mathbf{K}$ `false.`

Assume for definiteness that the above `ALL` expression is constructed as a result of the interpretation of the sentence *"Short women are humorous."* Suppose also that the expressions `(is-short Mary)` and `(is-woman Mary)` have been constructed. Then a few simple inference steps suffice to construct the expression `is-humorous Mary`. The inferencing can be done in many different ways. Barnden & Srinivas (1992b) describe one way for the sake of example. One of the inference rules it uses is a form of Universal Instantiation. This rule takes as input an expression of form `ALL(`a`)` and an expression b, and constructs ab. In our example situation, b would be `Mary` and a would be

$\mathbf{B}_3$ `IMPLIES (`$\mathbf{B}_3$ `AND is-short is-woman) is-humorous.`

ab is therefore

$\mathbf{B}_3$ `IMPLIES (`$\mathbf{B}_3$ `AND is-short is-woman) is-humorous Mary.`

After a reduction step on the outer $\mathbf{B}_3$, we get

`IMPLIES (`$\mathbf{B}_3$ `AND is-short is-woman Mary) (is-humorous Mary).`

Another $\mathbf{B}_3$ reduction then gives

`IMPLIES (AND (is-short Mary) (is-woman Mary))   (is-humorous Mary).`

This just says that if Mary is short and a woman then she is humorous. Barnden & Srinivas (1992b) spell out how a Conjunction Introduction rule and a Modus Ponens rule can use this `IMPLIES` expression and the existing individual expressions `(is-short Mary)` and `(is-woman Mary)` to construct the result expression `(is-humorous Mary)`.

Our intent here is not to detail how a realistic system could perform logical inferences in combinator form, but rather to discuss issues analogous to those that variables raise in standard logic. One of the most important issues is the question of what corresponds to variable substitution. The answer is that *it is "dissolved" into the more general notion of function application and the atendant combinator reductions.* The Universal Instantiation operation performed above merely involved the reduction of the function application ab, with a and b as specified. This corresponds in standard logic to taking the expression corresponding to a, namely `is-short(`x`)` $\wedge$ `is-woman(`x`)` $\Rightarrow$ `is-humorous(`x`)`, and within it substituting the constant `Mary` for x.

We now go on to consider more complex cases.

3.4 More Complex Cases of Representation and Substitution

The combinator expression for "Short women are clever" contained nothing that directly corresponded to the variable in the conventional logical formulation. However, other cases do introduce such a corresponding item. Consider the conventional-style formula

($\forall y$) `chase(John,` y`, angrily)` $\Rightarrow$ `flee(`y`, John, nervously).`

One combinator expression that is equivalent to the body of this formula is

(E_1) $\mathbf{B}_3$ `IMPLIES`

($\mathbf{B}_4$ `chase` ($\mathbf{K}$ `John`) $\mathbf{I}$ ($\mathbf{K}$ `angrily`))

($\mathbf{B}_4$ `flee` $\mathbf{I}$ ($\mathbf{K}$ `John`) ($\mathbf{K}$ `nervously`)).

Here the combinator $\mathbf{I}$ (the identity function) corresponds directly to the variable y. The $\mathbf{I}$ occurrences "capture" any argument expression that expression E_1 is applied to. For instance, suppose we apply E_1 to `Bill`. A reduction step on the $\mathbf{B}_3$ yields

`IMPLIES`

($\mathbf{B}_4$ `chase` ($\mathbf{K}$ `John`) $\mathbf{I}$ ($\mathbf{K}$ `angrily`) `Bill`)

($\mathbf{B}_4$ `flee` $\mathbf{I}$ ($\mathbf{K}$ `John`) ($\mathbf{K}$ `nervously`) `Bill`).

Reduction steps on the $\mathbf{B}_4$ occurrences then produce

(E_2) `IMPLIES (chase John Bill angrily) (flee Bill John nervously).`

The $\mathbf{B}_n$ occurrences distribute the argument `Bill` down to subexpressions. The **I** occurrences capture `Bill` and are replaced by just that constant. The **K** applications to `John`, `angrily` and `nervously` yield constant functions which serve to "cancel" `Bill`. For instance, **K** `angrily Bill` reduces just to `angrily`.

When we look at what corresponds to conventional logical formulae that use more than one variable, we find more complicated combinator subexpressions corresponding to variables. Consider the formula

$(\forall x, y)$ `chase(`x`,` y`, angrily)` $\Rightarrow$ `flee(`y`,` x`, nervously).`

A combinator expression equivalent to the body of this is:

(E_3) $\mathbf{B}_3^2$ `IMPLIES`

($\mathbf{B}_4^2$ `chase` **K** (**KI**) ($\mathbf{K}^2$ `angrily`))

($\mathbf{B}_4^2$ `flee` (**KI**) **K** ($\mathbf{K}^2$ `nervously`)).[3]

It is straightforward to verify that E_3`John` reduces to E_1, so that E_3`John Bill` reduces to E_2.

This example shows that when the conventional formulation has two variables, the combinator subexpression corresponding to one variable is **K**, and the subexpression corresponding to the other variable is **KI**. For three variables, it turns out that we get **BKK** (i.e. $\mathbf{K}^2$), **KK**, and $\mathbf{K}^2\mathbf{I}$ instead. *Thus, the combinator expression replacing any given variable is dependent on the total number of variables.*

In general, it turns out that the *ith* variable in a conventional logical formula (under some arbitrary ordering of the variables), in which there are j further variables, can be replaced by a special combinator expression which we abbreviate as $\mathbf{P}_{i,j}$. Notice carefully that we are counting *variables* here, not variable *occurrences.* A given variable may have several occurrences, and they will all be replaced by the same place-holder. The simplest way to define $\mathbf{P}_{i,j}$ is as follows:

$\mathbf{P}_{i,j}$ is $\mathbf{K}^{i-1}(\mathbf{K}^j$ (for $i > 0$ and $j \geq 0$).

[3] $\mathbf{B}_3^2$ and $\mathbf{B}_4^2$ are the squares of $\mathbf{B}_3$ and $\mathbf{B}_4$ respectively. Recall section 3.2 on powers of combinators. $\mathbf{B}_3^2 x$ means $\mathbf{B}_3(\mathbf{B}_3 x)$, not $(\mathbf{B}_3\mathbf{B}_3)x$.

However, since $\mathbf{K}^0$ is effectively just $\mathbf{I}$ (see section 3.2), an effectively equivalent definition leading to simpler expressions is as follows:

$\mathbf{P}_{1,0}$ is $\mathbf{I}$

$\mathbf{P}_{1,j}$ for $j > 0$ is $\mathbf{K}^j$.

$\mathbf{P}_{i,j}$ for $i > 1$ is $\mathbf{K}^{i-1}(\mathbf{P}_{1,j})$

However, the exact form of the $\mathbf{P}_{i,j}$ is not important for an understanding of the rest of the present paper. In examples we will not go beyond two variables. We refer to the special expressions $\mathbf{P}_{i,j}$ as "place-holders."

As an example, the place-holders for five variables, i.e. $\mathbf{P}_{1,4}$ to $\mathbf{P}_{5,0}$, are:

$\mathbf{K}^4$, $\mathbf{K}(\mathbf{K}^3)$, $\mathbf{K}^2(\mathbf{K}^2)$, $\mathbf{K}^3(\mathbf{K})$, $\mathbf{K}^4(\mathbf{I})$.

Additionally, the non-variable terms must "cancel" arguments by being surrounded by applications of $\mathbf{K}$. This is exemplified by $\mathbf{K}^2$`angrily` and $\mathbf{K}^2$`nervously` subexpressions in expression (E_3). When we wish to get the effect of a conventional formula with N variables, each constant c such as `angrily` must be surrounded by N nested applications of $\mathbf{K}$. That is, we must have $\mathbf{K}^N c$. We call $\mathbf{K}^N$ the "N-layer canceller." We call $\mathbf{K}^N c$ a "layered constant."

Sometimes, combinator expressions simpler than the ones we have presented can be used as analogs of logic expressions with variables. For instance, the combinator expression at the start of section 3.3 has no place holders, even though it corresponds to a logic formula of one variable. However, in the rest of the paper we will assume that the non-simplified forms are used, to abbreviate the discussion. Also, we do not claim that our place-holder scheme is the only or best combinator-based way of getting the effect of variables.[4]

[4]An alternative scheme is to have, instead of $\mathbf{P}_{i,j}$, the place-holder $\mathbf{Q}_i$, defined as $\mathbf{K}^{i-1}\mathbf{I}$. Although these place-holders are simpler, the analog of variable substitution becomes more complex, in a way mentioned below.

3.5 The Dissolving of Substitution

We now show more generally how it is that the substitution of expressions for variables is captured in our framework largely by *ordinary applications and reductions of combinator expressions.*

When $N > 0$, the application of a whole combinator expression E to an argument *that does not itself involve place-holders* automatically replaces the place-holders $\mathbf{P}_{i,j}$, cancellers $\mathbf{K}^N$ and distributors $\mathbf{B}_n^N$ in E by the ones needed for the effect of $N-1$ variables. This is because, for any argument a,

(i) $\mathbf{P}_{i,j}a$ for $i > 1$ reduces to $\mathbf{P}_{i-1,j}$.

(ii) $\mathbf{P}_{1,N-1}a$ is or reduces to $\mathbf{K}^{N-1}a$ (just a when $N = 1$).

(iii) $\mathbf{K}^N ca$ reduces to $\mathbf{K}^{N-1}c$ (just c when $N = 1$).

(iv) $\mathbf{B}_n^N fe_1\ldots e_{n-1}a$ for $N > 1$ reduces to $\mathbf{B}_n^{N-1}f(e_1a)\ldots(e_{n-1}a)$ (just $f(e_1a)\ldots(e_{n-1}a)$ when $N = 1$).

Point (i) shows the proper replacement of one place-holder by another (corresponding to leaving the non-first variables alone). Point (ii) shows the proper replacement of the first place-holder by an $(N-1)$-layer canceller for a (corresponding to substituting a for the first variable). Point (iii) shows the removal of one layer of cancellation for a constant c. Point (iv) shows a $\mathbf{B}_n^N$ doing a distribution action and replacing itself by the appropriate distributor for $N-1$ "variables." (The first section of the Appendix establishes points (i) to (iv) in detail.)

Thus, the reduction of Ea is analogous to substituting (the conventional-logic version of) a for the "first" variable of the conventional-logic version of E. By the first variable we mean the one corresponding to the "first" place-holder in E, namely $\mathbf{P}_{1,N-1}$.

By repetition, when E has N different place holders, the reduction of the multiple function application $Ea_1a_2\ldots a_k$ (for $1 \leq k \leq N$), where the a_i do not include place-holders, is analogous to substituting (the conventional-logic versions of) $a_1, \ldots, a_k$ for the first k variables in a standard logical formula. These variables are the ones corresponding to the place-holders $\mathbf{P}_{1,N-1}$ to $\mathbf{P}_{k,N-k}$.

Notice that E does not need to include all of the place-holders $\mathbf{P}_{1,N-1}$ to $\mathbf{P}_{N,0}$ for this "substitution" scheme to work. If $\mathbf{P}_{1,N-1}$ is missing, then the reduction of Ea leads simply to a slimmed down version of E — slimmed down in having its place-holders, distributors and cancellers slimmed down as in points (i), (iii), (iv).

We have seen that important types of variable substitution are "dissolved" in the combinator framework into ordinary function applications and expression reductions. However, to get the effect of conventional-logic substitutions of variables other than the first k variables for some k, or of substitutions where some substituting term itself is a variable or includes variables, we must go to somewhat more trouble if we wish to preserve the "dissolving" property (rather than just handling place-holders as if they were atomic variables).

The extra trouble is needed because if a "substituting" expression a contains place-holders, then Ea after reduction will have distributors $\mathbf{B}_{\cdot}^{N-1}$ and cancellers $\mathbf{K}^{N-1}$ — just as above — that are suitable for an expression with only $N-1$ "variables": but (as long as the place-holders in a are not among those of E) we want the ditributors and cancellers to be suitable for an expression with more than $N-1$ variables. Repeated "substitutions" will further slim down the distributors and cancellers. Also, point (ii) will, by itself, not have the right effect when when a is, say, just a place-holder $\mathbf{P}_{i,j}$ of E, because $\mathbf{P}_{1,N-1}a$ is $\mathbf{K}^{N-1}a$, which is a rewriting of $\mathbf{P}_{N-1+i,j}$, which is not a place-holder of E. Only after $N-1$ further "substitutions" will this place-holder be converted to $\mathbf{P}_{i,j}$.

One method which does deal with all cases of substitution is as follows. Let the "place-holder degree" of an expression be the value of $i+j$ for the place-holders $\mathbf{P}_{i,j}$ within it. (This value must be the same for all place holders.) Suppose that some of the "variables" are to be "substituted" by expressions a_i for some selection of values i in the range 1 to N, where the a_i may include place-holders, and where these place-holders may or may not be among those of E. Let N be the place-holder degree of E, and let M be the place-holder degree of the expression which is to result after the desired "substitutions." Then proceed as follows:

(1) For each j in the range 1 to N that is not one of the selected values i, let a_j be $\mathbf{P}_{j,M-j}$.

(2) Reduce $Ea_1a_2...a_N$, getting an expression F. Note that this is the desired expression except that its cancellers and distributors have all disappeared by virtue of repeated operation of points (iii) and (iv). What is correct in F is that in place of any place-holder $\mathbf{P}_{K,N-k}$ of E we now have a_k. (In particular, for the values j in step (1), if $M=N$ then $\mathbf{P}_{j,N-j}$ is replaced by itself, i.e. left unchanged.)

(3) Reintroduce distributors $\mathbf{B}_{\cdot}^{M}$ and cancellers $\mathbf{K}^{M}$ in the appropriate positions.

So, substitution is still largely dissolved into function application and expression reduction, but step (3) is an extra measure that counts against the purity of the dissolving.[5]

3.6 Other Operations on Variables

In order to conduct deduction in classical logic, operations on variables other than substitution are needed. In fact, it is possible to get by with only one other type of operation, namely generalization: the replacement of some or all occurrences of a particular constant in an expression by a variable that is not already in the expression (and the addition of an existential or universal quantification over that variable). Substitution and generalization are the only operations on variables needed in, for instance, the natural-deduction scheme presented in Davis (1990).[6]

It would be straightforward, though somewhat cumbersome, to tack an analog of generalization onto a combinator framework. Let the affected expression E have place-holders $\mathbf{P}_{1,N-1}$ to $\mathbf{P}_{N,0}$, and let the targeted constant be c. Generalization over some occurrences of c would include:

(1) replacing those place-holders by $\mathbf{P}_{1,N}$ to $\mathbf{P}_{N,1}$ respectively,

(2) adding an extra layer of cancellation to each layered constant in E (other than the occurrences of $\mathbf{K}^N c$ that are to be replaced),

(3) and changing each $\mathbf{B}_k^N$ in E to $\mathbf{B}_k^{N+1}$. (If $N = 0$, then distributors of form $\mathbf{B}_.$ need to be introduced.)

These steps are needed to get the effect of increasing the number of variables by one. Finally, the core of the generalization would consist of replacing the targeted occurrences of layered constant $\mathbf{K}^N c$ by $\mathbf{P}_{N+1,0}$.

This last operation is actually very simple. $\mathbf{P}_{N+1,0}$ is just $\mathbf{K}^N(\mathbf{I})$, so that the operation is just a matter of replacing c by $\mathbf{I}$. Also, step (1) is simple because $\mathbf{P}_{i,j}$ is (in general) $\mathbf{K}^{i-1}(\mathbf{K}^j)$, so that the step merely needs to replace $\mathbf{K}^j$ by $\mathbf{K}^{j+1}$. However, it must be said that the generalization process is not dissolved into other operations in the sense that substitution is.[7]

[5]The previous footnote mentioned an alternative place-holder scheme, using expressions $\mathbf{Q}_i$. Under this scheme, substitution works much as with the $\mathbf{P}_{i,j}$, but instead of substituend a_i we must instead use $\mathbf{K}^{N-i}a_i$.

[6]Of course, deduction is not the only type of inference one might wish to use over expressions that have compact generality. We have not investigated the application of the combinator framework to non-deductive inference.

[7]The alternative place-holder scheme using expressions $\mathbf{Q}_i$ makes generalization slightly simpler, as step (1) is not needed.

Another style of logical deduction is to base it on resolution operations, at whose center is the unification operation. It would be possible to take a standard unification algorithm and construct a straightforward analog of for combinator expressions, again essentially just by treating place holders as atmoic variables. However, in Barnden & Srinivas (1995:Appendix) we present an alternative unification algorithm that is more elegant in relying less heavily on a variable-like handling of place-holders, thereby dissolving into function-application and reduction to a greater extent.

3.7 Combinators in Connectionism

The ability to straightforwardly implement combinators in connectionism is not in doubt. Various connectionist frameworks have been developed that can rather directly encode and manipulate complex symbolic expressions (as short-lived representations). Letting these expressions be combinator expressions poses no special difficulties in principle. For instance, short-lived combinator expressions and the necessary manipulations (mainly reductions like those above) could readily be implemented either in BoltzCONS (Touretzky 1990) or in our own Conposit framework (Barnden 1991, Barnden & Srinivas 1991). Barnden & Srinivas (1992b) presents some of the necessary detail in these cases. Indeed, we have developed a symbolic combinator manipulation program, CONSKIM, using the symbolic representational and processing facilities supported by BoltzCONS, although that program does not perform inference of the sort discussed in the present paper. Other possibilities for future investigation include implementations of combinators in the connectionist frameworks of Lange & Dyer (1989), Shastri & Ajjanagadde (1993), Miyata, Smolensky & Legendre (1993), and Sun (1994).

By way of example, a natural implementation of combinator expressions in BoltzCONS would be as follows. The fundamental data structuring element in BoltzCONS is, at an abstract level of description, a 3-tuple of the form <**Tag**, **CAR**, **CDR**> fields. Each of the three components is a symbol. The **Tag** is used as a name of the tuple. A CAR or CDR symbol can either be the tag of some tuple or some symbol representing a basic datum. In the former case, the CAR or CDR field is to be interpreted as pointing to the named tuple. For instance, if tuples $< a, b, c >$ and $< b, d, e >$ are present, then the CAR field of the first points to the second tuple. Tuples are connectionistically encoded using a coarse, distributed encoding scheme over a large pool of units.

The details can be found in Touretzky (1990).[8] Now consider the combinator expression

`ALL(`$\mathbf{B}_3$` IMPLIES (`$\mathbf{B}_3$` AND is-short is-woman) is-humorous).`

With parentheses inserted to make explicit the implied grouping of subexpressions, the expression is

`ALL(((`$\mathbf{B}_3$` IMPLIES) (((`$\mathbf{B}_3$` AND) is-short) is-woman)) is-humorous).`

It can then be viewed as a binary tree, as shown in Figure 2. The internal nodes represent function applications. This tree can in turn be represented as a set of tuples as shown in Figure 3. Each tuple in Figure 3 corresponds to an internal (i.e., non-leaf) node in Figure 2, and for convenience we have taken the tuple's tag to be the node label used in Figure 2. Notice how tuples "point" to each other by means of tags; for instance, the tuple with tag `t1` points to the tuple with tag `t2` via its CAR field. Barnden & Srinivas (1992b) outline how reduction operations could be done on tuple-based implementations of combinator expressions in BoltzCONS.

FIG. 2 ABOUT HERE

FIG. 3 ABOUT HERE

Another class of possibilities for importing combinators into connectionism, this time less directly, is exemplified by the RAAM method of Pollack (1990) and the encoding method of Plate (1991). These approaches encode complex symbolic structures as *compressed encodings*, vectors whose size is independent of the size of the structures. One aim within this paradigm is to be able to process the symbolic structures holistically, in the sense that the compressed encoding for a complex structure is manipulated as a whole, without the extraction of encodings of the constituents of the structure. In our case, the compressed encodings would encode combinator expressions, and two crucial processing operations would be the formation of an application of one expression to another and the reduction of such an application in the ways exemplified above. It is not yet clear, however, whether such reduction operations (and applications of inference rules such as those alluded to in section 3.3) can in fact adequately be done by holistic processing. The elaborateness and generality of the processing goes considerably beyond what has actually been demonstrated to be achievable with compressed encodings, in such works as Blank *et al.*

[8]Touretzky (1990) also describes a version of BoltzCONS based on 5-tuples. In the present paper we only require 3-tuples.

(1992), Chalmers (1990), Chrisman (1991), Niklasson & van Gelder (1994) and Pollack (1990). Nevertheless, the holistic compressed encoding approach to combinators will figure importantly in the discussion below.

3.8 Advantages of Combinators for Connectionism

Why might it be advantageous for a high-level connectionist system — one aimed at natural language understanding, common-sense reasoning, etc. — to implement or approximately emulate combinator expressions as opposed to more standard types of symbolic expression? The answer lies mainly in various types of *uniformity* that the combinator method supplies.

First, combinators provide greater *syntactic uniformity.* All expressions are just function applications or constants (where a constant can be a function name). By contrast, standard logical representation schemes have various different types of expression, depending on whether connectives, quantifiers, etc. are present. In addition, in the combinator framework all functions have one argument. The greater uniformity of combinator expressions makes them simpler to encode in a connectionist system.

As an added bonus there is is greater *uniformity of processing*: the variety of different symbol-manipulation operations that need to be implemented/approximated is reduced. This is a welcome simplification in itself, but as a side-effect it also reduces the overall control problem for the system. A non-combinator-based system might need one subnetwork to deal with variable substitutions and another to deal with function applications. This then raises the need for circuitry to steer inputs to these two subnetworks as appropriate. In a combinator-based system, the distinction would not need to be made.

There is a further potential benefit of the greater syntactic uniformity and the consequent processing uniformity. They make it easier to see how *parallelism* could be used within (approximations to) symbolic manipulations. Even if a system proceeds serially at a high-level of description, it could be beneficial to implement a given step in the series in a parallel way in order to speed that step up. For instance, consider reducing the application of a complex combinator expression like those above — involving place-holders, cancellers and powers of $\mathbf{B}_n$ — to some argument expression. The $\mathbf{B}_n$ combinators serve to distribute the argument down to subexpressions, which are then applied to the argument. These different applications could be farmed out to modules working in parallel. An analogous farming-out could be done in a variable substitution operation on a standard logical expression, but in the combinator framework the individual modules are simpler and the communication between the overall steering process and the individual modules is simpler. The only types of expression communicated are atomic symbols and single-argument function applications. As one result of these simplifications, modules could gain efficiency through not having to be so concerned with distinguishing between different types of syntactic structure.

Apart from the above syntactic uniformity and processing uniformity, there is another way in which uniformity of expressions is increased. Combinators provide an extra degree of *canonicalization.* Standard logical frameworks have to be able to cope with the fact that two expressions S_1 and S_2 might differ only through a consistent renaming of their variables. However, given a certain assumption, the corresponding combinator expressions do not differ in any analogous respect — they use the same place-holders, and are therefore the same expression C. The mentioned assumption is that the place-holders $\mathbf{P}_{1,N-1}$, $\mathbf{P}_{2,N-2}$, ..., $\mathbf{P}_{N,0}$ in a combinator expression correspong to a conventional logic expression with N variables occur in some canonical order. A natural canonical order is for the first occurrence of $\mathbf{P}_{m,N-m}$ to precede the first occurrence of $\mathbf{P}_{n,N-n}$ when $m < n$.

It could be important for a cognitive system to notice that it is entertaining essentially-identical expressions like S_1 and S_2. For instance, if two different chains of inference came up with these expressions, it could be important for the system to notice the "identity" so as to boost its level of confidence in the conclusion. In a combinator version of such a system, we have the simpler problem of noticing the identity of C with itself.

However, we have to be a little careful here. Consider first the case in which combinator expressions are implemented as compressed encodings. To notice the identity of a combinator expression C with itself, the system only needs to compare the compressed encoding of C with itself – an easy vector equality detection. By contrast, certain other ways of implementing combinator expressions in connectionism would essentially reintroduce variables by the back door, undoing the possibility of an identity-match advantage. This happens, for instance, in the approach to implementing combinator expressions in BoltzCONS that was specified above. The expressions are cashed out as symbol tuples, associatively connected together by the sharing of arbitrarily chosen "tag" symbols. The tags can be viewed as a type of variable. Two different, simultaneous realizations of the very same combinator expression would have to use different tags.[9]

[9]The same phenomenon would arise in a Conposit implementation, assuming the version of Conposit we described

A possible objection to applauding the combinator approach for its canonicalization is that a standard symbolic approach could itself use variables canonically. Assume that the system's variables form an ordered set, $[x_1, x_2, x_3, \ldots]$. Then, in any expression with n variables, those variables can be constrained to be x_1 to x_n, and their occurrences can be put in numerical order (i.e., the first occurrence of x_1 is before the first occurrence of x_2, etc.). This is similar to the canonical ordering of place-holders that we suggested above, and we see that place holders are strongly analogous to canonical variables. Canonical variables lead to the same identity-matching advantage as was described a moment ago.

However, the operation of substituting for the first variable some term not containing variables will leave the expression in a non-canonical state. Therefore, the substitution must be followed by an extra operation of re-canonicalization. By contrast, as we saw in section 3.5, the place holders and cancellers in a combinator expression are *automatically* readjusted in the course of doing the analogue of a variable substitution operation, whn the substituends do not contain variables. (And the canonical variable approach does not have the syntactic and processing uniformity advantage discussed above.)

Combinators have long been proposed and used for the implementation of some "functional" programming languages, in order to eliminate variables (Turner 1979; Peyton Jones 1987). The main benefits of this are more efficient implementation (Turner 1979; Hudak & Goldberg 1985), and the avoidance of side-effects, leading to computational determinacy properties that are of great potential use in parallel processing (Peyton Jones, Clack, & Salkid 1989). However, the parallel processing advantage involved in this line of work concerns long-lived procedures encoded as combinator expressions, so that it is not directly relevant to the bulk of the present paper, with its focus on short-lived expressions.

3.9 Disadvantages of Combinators for Connectionism

The syntactic uniformity advantage of combinators is a matter of decreased qualitative complexity. However, combinators lead to increased *quantitative* complexity — the size of expressions is significantly increased. This can be seen, for instance, from the combinator expression (E_3) above. The

in Barnden (1991). The implementation involves symbols highly analogous to the BoltzCONS tags (see Barnden & Srinivas 1991 on this point). However, in later work (Barnden & Srinivas 1992a), we replaced the arbitrary tag-like symbols by compressed encodings that are the same on all occasions of implementing a given symbolic structure. This technique regains the ability to do rapid identity matching.

extra size comes from the $\mathbf{B}_n^N$ occurrences, where N is the number of "variables," the place-holders $\mathbf{P}_{i,N-1}$ and the cancellers $\mathbf{K}^N$.

These extra levels may lead to more processing steps overall, other things being equal. However, the increase could be compensated for by the speed-from-parallelism advantage noted in the previous subsection. Of course, that could mean that the parallelism advantage could largely be used up by increased quantitative complexity, not leaving much overall benefit.

In fairness, we should not forget that having variables can itself lead to extra processing steps. For instance, we mentioned re-canonicalization of variables in section 3.8.

An increased size of symbolic expressions could be a special disadvantage in a compressed-encoding connectionist combinator system. The increase could be reflected in increased degradation of the compressed encodings. For instance, in a RAAM system where the compression mechanism is binary (i.e. a combinator expression is viewed as a binary tree, with an internal node for each function application), the distributors, place holders and cancellers contribute added depth to the tree. The increased depth results in more compression operations to get the encoding for the overall structure, increasing the opportunity for encoding errors to arise.

Finally, combinators introduce a type of syntactic *non*-uniformity, in that the choice of cancellers and distributors depends on the total number N of "variables" in an expression. A possible negative effect of this non-uniformity is mentioned at the end of section 4.

4 Further Discussion

If combinators were to be used in connectionism, it would be because of benefits such as those in section 3.8, not because of the sheer fact that they avoid variables. However, it is natural to look at other conceivable variable-avoidance techniques to see whether they have similar advantages or others. An extended treatment of this matter is beyond the scope of the present paper, but one or two tentative points can be made about the non-combinator variable-avoidance possibilities mentioned in section 2.

The two natural-language-based techniques (sections 2.1 and 2.2) could, if properly regimented, provide some degree of canonicalization — for example, phrases like "the second person" do not require the use of arbitrarily chosen variable names. On the other hand, none of the techniques of section 2 seem to provide the fine-grained type of syntactic and processing uniformity achievable with combinators. (In particular, in the semantic network approach ofsection 2.4, the analogue of variable substitution is still a special operation, rather than being "dissolved" into some other operation. Recall that, in the combinator framework, variable substitution is dissolved into ordinary function application and attendant reduction activity). Thus, semantic networks do not have the particular simplicity and parallelism advantages noted for combinator expressions in section 3.8.

A further question is whether the semantic network approach could exhibit something like the canonicalization that can be provided by combinators section 3.8. In fact, despite the lack of overt variable names, it would not provide canonicalization unless a somewhat subtle type of encoding of the networks were used. Suppose we simply encoded networks on the basis of a mapping of nodes to particular connectionist activation patterns or to particular connectionist units. This could result in non-canonical encodings, because of different choices of connectionist pattern or unit on different occasions of encoding the same semantic network fragment. To avoid this, we could devise a systematic scheme for saying which variable nodes were the first, second, third, and so on in the network fragment, and then always mapping the *ith* variable node in any fragment, for any given i, into a standard connectionist unit or activation pattern.

However, canonicalization could conceivably arise automatically in a system like DUAL (Dyer, Flowers and Wang 1988; see also Dyer, 1991). Here, semantic network nodes are connectionistically realized as compressed encodings arising from a cyclic, inter-node cooperative process. This opens up the possibility of a given semantic network fragment always being encoded in the same way. However, further research is needed to establish whether this would in fact happen.

As regards canonicalization in the mental model approach (section 2.3), a difficulty is that Johnson-Laird's theory allows the number of tokens used in models to vary from occasion to occasion, perhaps randomly (Johnson-Laird 1983, Johnson-Laird & Byrne 1991). If mental models were to be encoded as holistically manipulated compressed encodings, this is a degree of arbitrariness that might be undesirable, detracting from the benefit of eliminating variables. In a compressed-encoding approach (section 3.7) one might therefore wish to standardize the numbers of tokens used per category. Similarly, the tokens might need to be encoded as *canonical* activation patterns. In that case, there would be activation patterns A_1, A_2, ... A_N for some N; and in problem solving involving k categories each token of the first category encountered in the problem would be encoded by means of A_1, each token of the second category would be encoded by A_2, and so on.

Finally, one matter on which we are far from clear is whether high-level connectionist systems would be able to learn to do their inferencing more easily if they used combinators than if they used variables. Barnden & Srinivas (1992b) present some reason for thinking this may be the case, at least for connectionist systems doing holistic processing of compressed encodings. On the other hand, the non-uniformity noted at the end of section 3.9 could make it harder for a learning system to be sensitive to important similarities between expressions that happened to have different numbers of "variables."

5 Conclusion

We have argued out that connectionist concern over variables should be expanded to a concern over compact generality. There should also be more attention to compact generality in short-lived items such as encodings of natural language inputs as opposed to long-lived items such as rules in long-term memory. A special feature of connectionist encoding practices is that long-term information is usually encoded in a very different way (*viz.* as weights) from short-lived information (*viz.* as activation patterns). Therefore, variable-handling techniques (or compact generality techniques more generally) that are used for long-lived items do not transfer readily to short-lived items, or vice versa.

The shift from the notion of variable-based quantification to the notion of compact generality is a genuinely useful one, because there is at least one well developed variable-free paradigm for compact generality, namely the combinator approach. We saw that combinators have certain benefits and drawbacks as targets for implementation or approximate emulation in connectionism. The benefits lie in added uniformity of syntax and processing, leading to simpler connectionist implementation/emulation and greater parallelizability, and in greater canonicalization, leading to possible connectionist optimizations in the matching of complex short-lived structures. Notice also that canonicalization of short-lived representations connects to the issue of systematicity of inference. The more canonical the representations, the smaller the space of systematically different representations that a given processing mechanism must deal with.

The drawbacks of combinators lie in bigger expressions and in the fact that some operations become more expensive. It is therefore unclear whether or not variables should be abandoned in favor of combinators.

TO summarize the consequences of the study of combinator approaches for cognitive science: although we do not aspire to provide a combinator-based cognitive model and do not claim that combinators have psychological reality, our study addresses general issues that could inform the creation of future cognitive models. Specifically, the thrust is (i) to guard against a rejection of symbolic notions on the grounds that they necessarily bring variables, and (ii) to underscire the importance of compact generality within short-lived representations, not just in rules(the usual locus of discussion in debates concerning variables). Both of these issues are, we believe, of considerable importance to cognitive science. We should also note that, although this paper has concentrated on deductive reasoning, the basic issues also arise in other types of reasoning, such as default reasoning, induction, abduction, and some forms of analogy-based and case-based reasoning. Thus, the potential value of considering methods for avoiding variables in short-lived representations is broad.

ACKNOWLEDGMENTS

We are grateful to Tom Eskridge, Gopal Gupta and anonymous referees for useful comments.

APPENDIX

Adequacy of Place-Holder Scheme

If points (i) to (iv) in section 3.5 are valid, then the presented place-holder scheme provides a valid analog of variables, by the following argument. We will consider only conventional logic expressions in prenex form, i.e. with all quantifiers at the front(all conventional logic formulae can be converted to prenex form). We will be concerned only with the bodies of such expressions, i.e. the portion after the quantifiers.

Suppose an expression E is the analog of conventional logic expression E' that contains N different variables (but no quantifiers), and that E has place-holders $\mathbf{P}_{1,N-1}$ to $\mathbf{P}_{N,0}$. We will say that the variable corresponding to $\mathbf{P}_{i,j}$ is the *ith* variable. Let expressions a_1 to a_N be analogues of conventional logic terms a'_1 to a'_N that do not contain variables, so that a_1 to a_N do not contain place-holders. We wish to show that $Ea_1 \dots a_N$ reduces to the combinator analog of the conventional expression resulting from substituting a_i for the ith variable of E for each i.

By repeated application of points (i) to (iv) it is easy to see that $Ea_1 \dots a_N$ reduces to an expression F in which all occurrences of place-holder$\mathbf{P}_{i,j}$ of E have been replaced by a_i, and in which there are no distributors or cancellers. Then, since the set of occurrences of $\mathbf{P}_{i,j}$ correspond in position to the set of occurrences in E' of the *ith* variable, F is just the analog of the expression resulting from substituting a'_i for the *ith* variable in E', for each i. This shows that substitution interacts correctly with the analog relationship between conventional and combinator expressions. It remains to prove points (i) to (iv).

Point (i):

Suppose $i > 1$. Then $\mathbf{P}_{i,j}a$ is $\mathbf{K}^{i-1}\mathbf{P}_{1,j}a$, which can be rewritten as $\mathbf{K}(\mathbf{K}^{i-2}\mathbf{P}_{1,j})a$, which reduces to $\mathbf{K}^{i-2}\mathbf{P}_{1,j}$, which is just $\mathbf{P}_{i-1,j}$, as desired. (In this argument, if $i = 2$ then $\mathbf{K}^{i-2}$ can be deleted.)

Point (ii):

Case when $N = 1$: $\mathbf{P}_{1,0}a$ is $\mathbf{I}a$, which reduces to a, as desired.

Case when $N > 1$: $\mathbf{P}_{1,N-1}a$ is $\mathbf{K}^{N-1}a$, as desired, directly from the definition of $\mathbf{P}_{1,j}$ when $j > 0$.

Point (iii):

Case when $N = 1$: $\mathbf{K}ca$ reduces to c, as desired.

Case when $N > 1$: $\mathbf{K}^N ca$ can be rewritten as $\mathbf{K(K}^{N-1}c\mathbf{)}a$, which reduces to $\mathbf{K}^{N-1}c$, as desired.

Point (iv):

Let $N > 1$. Then $\mathbf{B}_n^N f e_1 \ldots e_{n-1} a$ can be rewritten as $\mathbf{B}_n\mathbf{(B}_n^{N-1} f) e_1 \ldots e_{n-1} a$, which reduces to $\mathbf{B}_n^{N-1} f (e_1 a) \ldots (e_{n-1} a)$, as desired.

REFERENCES

Barnden, J.A. (1989). Neural-net implementation of complex symbol-processing in a mental model approach to syllogistic reasoning. In *Procs. 11th Int. Joint Conf. on Artificial Intelligence,* pp.568–573. San Mateo, CA: Morgan Kaufmann.

Barnden, J.A. (1991). Encoding complex symbolic data structures with some unusual connectionist techniques. *Advances in Connectionist and Neural Computation Theory, Vol. 1: High Level Connectionist Models,* pp.180–240. Norwood, N.J.: Ablex Publishing Corp.

Barnden, J.A. (1992). Connectionism, generalization and propositional attitudes: a catalogue of challenging issues. In J. Dinsmore (ed), *The Symbolic and Connectionist Paradigms: Closing the Gap.* Hillsdale, N.J.: Lawrence Erlbaum. pp.149–178.

Barnden, J.A. (1994). Complex symbol-processing in Conposit, a transiently localist connectionist architecture. In R. Sun & L. Bookman (Eds), *Computational Architectures for Integrating Neural and Symbolic Processes.* Kluwer.

Barnden, J.A. (1995). High-level reasoning, computational challenges for connectionism, and the Conposit solution. *Applied Intelligence, 5* (2), pp.1–33.

Barnden, J.A. & Srinivas, K. (1991). Encoding techniques for complex information structures in connectionist systems. *Connection Science, 3* (3), pp.263–309.

Barnden, J.A. & Srinivas, K. (1992a). Overcoming rule-based rigidity and connectionist limitations through massively-parallel case-based reasoning. *Int. J. Man-Machine Studies, 36*, pp.221–246.

Barnden, J.A. & Srinivas, K. (1992b). Working memory variables, logical combinators and systematicity. *Memoranda in Computer and Cognitive Science*, No. MCCS–92–245, Computing Research Laboratory, New Mexico State University, Las Cruces, NM 88003.

Barnden, J.A. & Srinivas, K. (1995). Combinators for Connectionism: Quanitification, Substitution and Unification without Variables *Memoranda in Computer and Cognitive Science*, No. MCCS–95–286, Computing Research Laboratory, New Mexico State University, Las Cruces, NM 88003.

Blank, D.S., Meeden, L.A. & Marshall, J.B. (1992). Exploring the symbolic/subsymbolic continuum: a case study of RAAM. In Dinsmore, J. (Ed.), *The Symbolic and Connectionist Paradigms: Closing the Gap*, pp. 113–148. Hillsdale, N.J.: Lawrence Erlbaum.

Chalmers, D.J. (1990). Syntactic transformations on distributed representations. *Connection Science, 2* (1 & 2), pp.53–62.

Chapman, D. & Agre, P.E. (1987). Abstract reasoning as emergent from concrete activity. In M.P. Georgeff & A.L. Lansky (Eds), *Reasoning about Actions and Plans.* Los Altos, CA: Morgan Kaufmann.

Chrisman, L. (1991). Learning recursive distributed representations for holistic computation. *Connection Science, 3* (4), pp.354–366.

Curry, H.B., & Feys, R. (1958). *Combinatory logic.* Amsterdam: North-Holland.

Davis, E. (1990). *Representations of commonsense knowledge.* San Mateo, CA: Morgan Kaufmann.

Dyer, M.G. (1991). Symbolic NeuroEngineering and natural language processing: a multilevel research approach. In J.A. Barnden & J.B. Pollack (Eds.), *Advances in Connectionist and Neural Computation Theory, Vol. 1.* Norwood, N.J.: Ablex Publishing Corp. pp.32–86.

Dyer, M.G., Flowers, M. & Wang, Y.A. (1988). Weight-matrix = pattern of activation: encoding semantic networks as distributed representations in DUAL, a PDP architecture. Tech. Report UCLA-AI-88-5, Department of Computer Science, University of California, Los Angeles, CA.

Fodor, J.A & Pylyshyn, Z.W. (1988). Connectionism and cognitive architecture: A critical analysis. In S. Pinker & J.Mehler (Eds.), *Connections and Symbols.* Cambridge, MA: MIT Press and Amsterdam: Elsevier.

Hadley, R.F. (1993a). Connectionism, explicit rules, and symbolic manipulation. *Mind and Machines, 3* (2), pp.183–200.

Hadley, R.F. (1994).Systematicity in connectionist language learning. *Mind and Machines, 9* (3).

Harris, C.L. & Elman, J.L. (1989). Representing variable information with simple recurrent networks. In *Procs. Eleventh Annual Conf. of the Cognitive Science Society,* pp.635–642. Hillsdale, N.J.: Lawrence Erlbaum.

Hinton, G.E. (1990). Mapping part-whole hierarchies into connectionist networks. *Artificial Intelligence, 46* (1–2), pp.47–75.

Hölldobler, S. (1990). A structured connectionist unification algorithm. In *Procs. 8th National Conf. on Artificial Intelligence (AAAI–90),* pp.587–593. Menlo Park, CA: AAAI Press.

Hudak, P. & Goldberg, B. (1985). Efficient distributed evaluation of functional programs using serial combinators. *IEEE Trans on Computers, C-34*, (10), pp.831–839.

Johnson-Laird, P.N. (1983). *Mental models: towards a cognitive science of language, inference and consciousness.* Cambridge, Mass.: Harvard University Press.

Johnson-Laird, P.N. & Byrne, R.M.J. (1991). *Deduction.* Hove, U.K.: Lawrence Erlbaum.

Lange, T.E. & Dyer, M.G. (1989). High-level inferencing in a connectionist network. *Connection Science, 1* (2), pp.181-217.

Miyata, Y., Smolensky, P. & Legendre, G. (1993). Distributed representation and parallel processing of recursive structures. In *Procs. 15th Annual Conf. of the Cognitive Science Society.* Hillsdale, N.J.: Lawrence Erlbaum.

Niklasson, L. & van Gelder, T. (1994). Can connectionist models exhibit non-classical structure sensitivity? In *Procs. Sixteenth Annual Conference of the Cognitive Science Society*, pp.664–669. Hillsdale, N.J.: Lawrence Erlbaum.

Peyton Jones, S.L. (1987). *The implementation of functional programming languages.* Englewood Cliffs, N.J.: Prentice-Hall.

Peyton Jones, S.L., Clack, C. & Salkid, J. (1989). High performance parallel graph reduction. In Proceedings of PARLE '89, Parallel Architectures and Languages Europe, pp. 193–206. Springer-Verlag.

Pinkas, G. (1992). Constructing proofs in symmetric networks. In J.E. Moody, S.J. Hanson, R.P. Lippmann (Eds.), *Advances in Neural Information Processing Systems*, pp. 217-224. San Mateo: Morgan Kaufmann.

Pinker, S. & Prince, A. (1988). On language and connectionism: analysis of a parallel distributed processing model of language acquisition. In S. Pinker & J. Mehler (Eds.), *Connections and symbols*, Cambridge, Mass.: MIT Press. (Reprinted from *Cognition, 28,* 1988.)

Plate, T. (1991). Holographic reduced representations: convolution algebra for compositional distributed representations. In *Procs. 12th Int. Joint Conf. on Artificial Intelligence* (Sydney, Australia, Aug. 1991), pp.30–35. San Mateo: Morgan Kaufmann.

Pollack, J.B. (1990). Recursive distributed representations. *Artificial Intelligence, 46* (1–2), pp. 77–105.

St. John, M.F. & McClelland, J.L. (1990). Learning and applying contextual constraints in sentence comprehension. *Artificial Intelligence, 46* (1-2), pp.217–257.

Schönfinkel, M. (1924). Uber die Bausteine der mathematischen Logik. *Math Ann., 92,* p.305.

Shastri, L. & Ajjanagadde, V. (1993). ¿From simple associations to systematic reasoning: a connectionist representation of rules, variables, and dynamic bindings using temporal synchrony. *Behavioral and Brain Sciences, 16* (3), pp.417–494.

Smolensky, P. (1988). On the proper treatment of connectionism. *Behavioral and Brain Sciences, 11*, 1-74.

Sun, R. (1994). *Integrating rules and connectionism for robust commonsense reasoning.* New York: Wiley.

Touretzky, D.S. (1990). BoltzCONS: dynamic symbol structures in a connectionist network. *Artificial Intelligence, 46* (1–2), pp. 5–46.

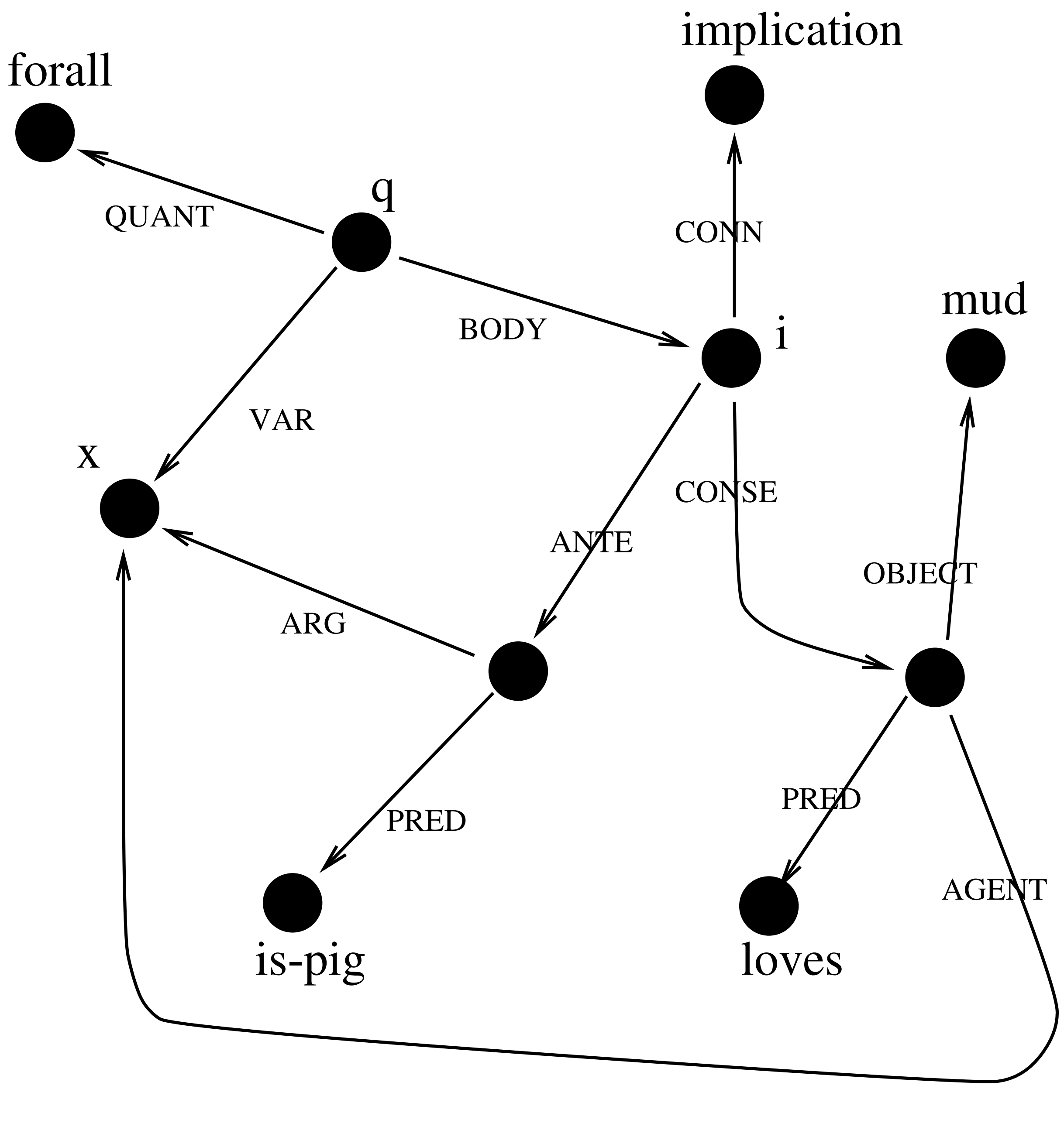
Turner, D.A. (1979). A new implementation technique for applicative languages. *Software Practice and Experience, 9*, pp.31–49.

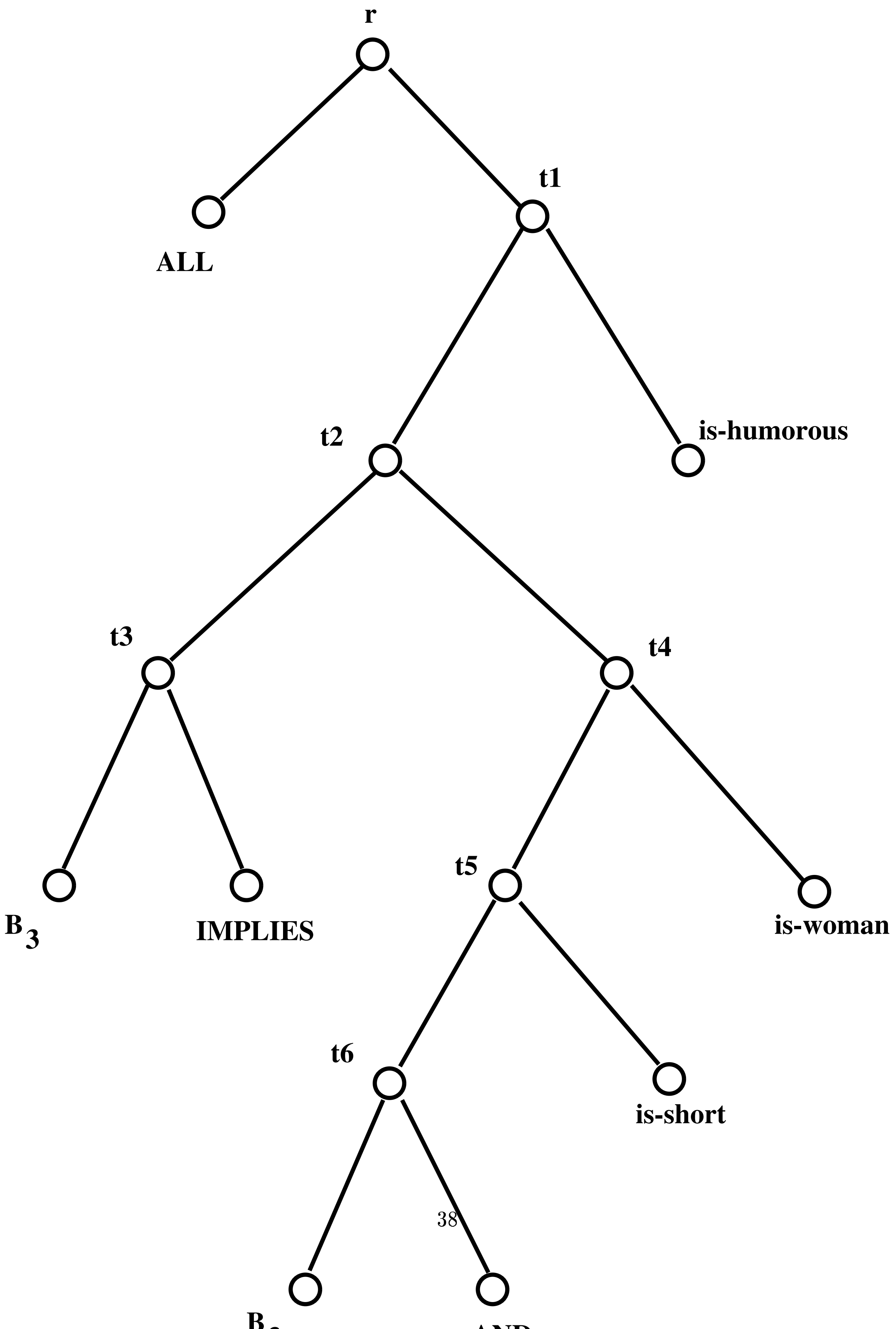
FIGURE CAPTIONS

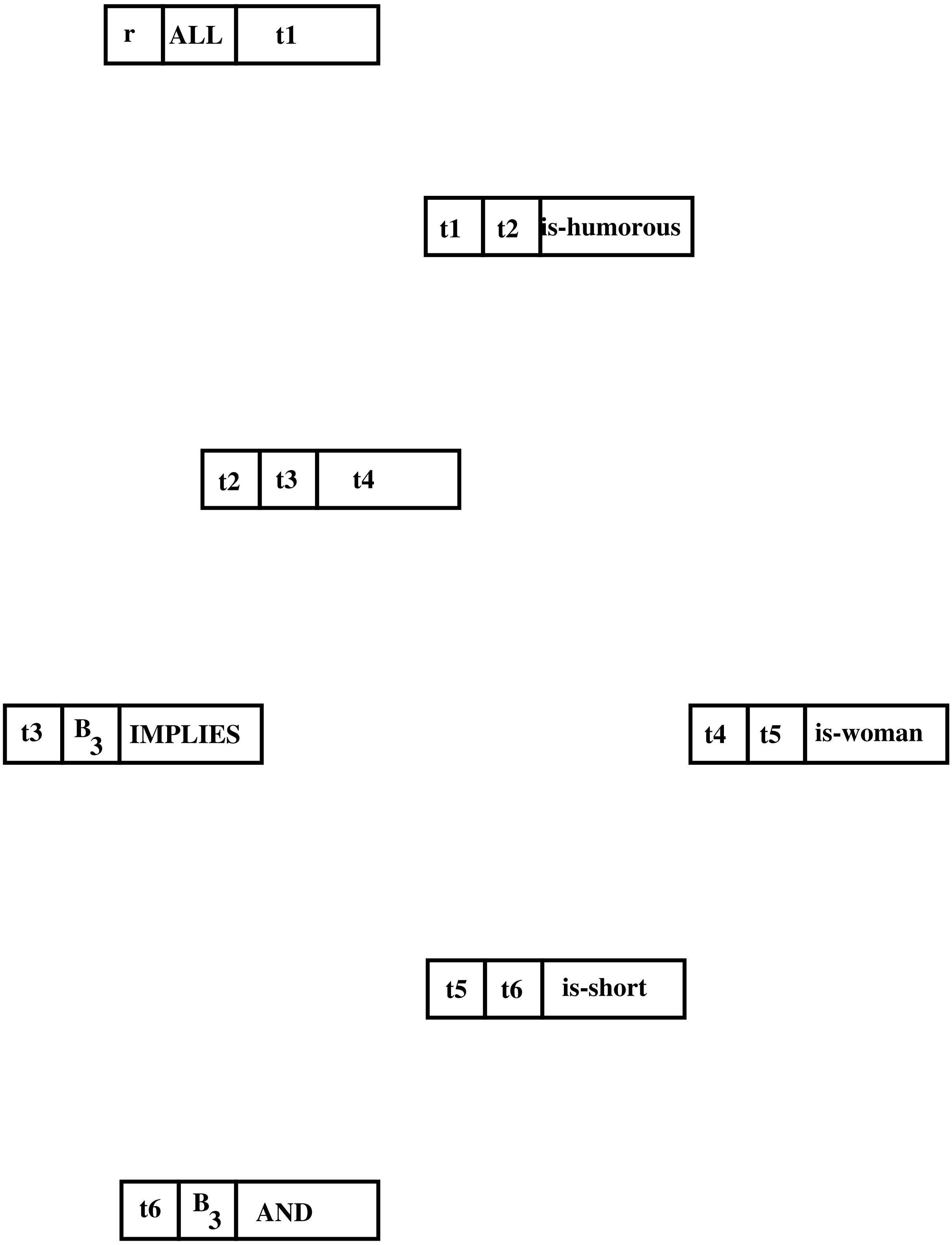
(1) A possible semantic network fragment expressing the proposition that all pigs love mud. The fragment is exactly analogous to the logic formula `(`$\forall x$`) is-pig(x)` $\Rightarrow$ `loves(x, mud).` The whole proposition is headed by the node q. The x node is analogous to the logic variable `x`. The three occurrences of `x` in the formula are replaced by the three links to the x node in the network. The label "x" in the figure is purely for purposes of reference in the text — the diagram would still be correct without it.

(2) A binary tree representation of a combinator expression in the text. Each internal (i.e., non-leaf) node represents the application of a function to an argument, with the function being represented by the left subtree of the node and the argument being represented by the right subtree. The symbols `r, t1, t2,` etc. are for reference only, and are not part of the tree as such.

(3) A set of tuples that a BoltzCONS system could use to represent the binary tree shown in Figure 2. Each box illustrates a tuple, where each tuple has a tag field, a CAR field and a CDR field going from left to right within the box. The CAR field can used be used to "point" to the left child (if any) of the correspondig tree node, and the CDR field can similarly be used for the right child (if any). There is one tuple for each internal node in Figure 2.


forall
implication
q
QUANT
CONN
BODY
i
mud
VAR
x
CONSE
ANTE
OBJECT
ARG
PRED
PRED
AGENT
is-pig
loves


r
ALL
t1
is-humorous
t2
t3
t4
B_3
IMPLIES
t5
is-woman
t6
is-short


r ALL t1
t1 t2 is-humorous
t2 t3 t4
t3 B_3 IMPLIES
t4 t5 is-woman
t5 t6 is-short
t6 B_3 AND