Journal of Psycholinguistic Research

How Language Could Have Evolved

--Manuscript Draft--

Manuscript Number:	JOPR-D-21-00230R5
Full Title:	How Language Could Have Evolved
Article Type:	Original Research
Keywords:	great leap theory of mind
Corresponding Author:	ken delsignore Case Western Reserve University North Aurora, IL UNITED STATES
Corresponding Author Secondary Information:	
Corresponding Author's Institution:	Case Western Reserve University
Corresponding Author's Secondary Institution:	
First Author:	ken delsignore
First Author Secondary Information:	
Order of Authors:	ken delsignore
Order of Authors Secondary Information:	
Funding Information:	
Abstract:	This paper begins to develop a biologically inspired computational model of the Human language faculty and some associated thought processes. This model is developed starting from a simple proto-language, which humans are assumed to have inherited at speciation. This proto- language consists of single symbol exchange using a small set of symbols; similar to the observed dialogue systems in the existing Great Ape families. Computationally, the model is built using a single class with the form of a Markov graph node. Instances of this node class are used to symbolically represent words. The model is built iteratively in main() as a single graph. Nodes are added to this graph using a merge, or conjunctive join, operation between any two existing nodes, notionally labeled as head and copy. A simple first order graph is developed which is hypothesized to be common to all Mammals and to generate shared mammal behaviours. This graph is then extended to allow for more complex human language and thought processes.

Powered by Editorial Manager® and ProduXion Manager® from Aries Systems Corporation

blank file

Title Page

How Language Could Have Evolved

Ken Del Signore
1509 Patterson Ave.
North Aurora, Il. USA
kendelsignore@gmail.com

Author Ken Del Suignore declares that he/she has no conflict of interest.

Manuscript blinded
Click here to access/download;Manuscript blinded;paper1 (3).docx
Click here to view linked References

How Language Could Have Evolved

Abstract

This paper begins to develop a biologically inspired computational model of the Human language faculty and some associated thought processes. This model is developed starting from a simple proto-language, which humans are assumed to have inherited at speciation. This proto-language consists of single symbol exchange using a small set of symbols; similar to the observed gestural communication systems in the existing Great Ape families. Computationally, the model is built using a single class with the form of a Markov graph node. Instances of this node class are used to symbolically represent words. The model is built iteratively in main() as a single graph. Nodes are added to this graph using a merge, or conjunctive join, operation between any two existing nodes, notionally labeled as head and copy. A simple first order graph is developed which is hypothesized to be common to all Mammals and to generate shared mammal behaviours. This graph is then extended to allow for more complex human language and thought processes.

Keywords: great leap theory of mind

1 Introduction

We take the simplest possible communication to be a single symbol, exchanged from one communicator to another. As an example, consider a hiker that is lost in the woods and builds a pile of rocks, and then moves on. If a second hiker subsequently finds the rocks, then we can say that a single symbol has been exchanged. The information conveyed is the same as if the first hiker had just stood next to the second and said the adverb “here”, except that the hiker said “here” some long time prior and the symbol (the rocks) held the information through time. There is no other temporal or spatial information conveyed.

To modify this example, if while the hiker is piling the rocks, s/he hears the second hiker coming straight on and calls out “here”, then the second hiker will have, at the simplest, one additional quanta of information, namely a measured value of the distance. This second quanta is analogous to a floating point variable that can take on a continuous value.

Single symbol dialogue systems are observed throughout the animal and plant kingdoms. The symbol “here” is often the exchanged symbol in these systems. A flower can be interpreted as a single symbol, exchanged between a plant and its pollinators, with the meaning of the flower being “here” (Chomsky 2015). A symbol is defined to contain two quanta of information: label and value; with value being possibly zero or unspecified. In the examples above, the second communicator measures the value locally to itself using the externalization made by the first communicator.

The neocortex gives mammals the ability to store and recall sequences of symbols with relative ease. This ability gives mammals considerably more complex behavior relative to non mammals. Mammals can input and store sequences of symbols in combinations never experienced before and later recall and utilize this information; an example of which would be the second hiker remembering the path out of the woods and walking out with the first hiker.

All known mammal dialogue, excluding human, uses or can be easily reduced to single symbol exchange. The great apes have possibly the most developed system; using a gestural vocabulary of approximately 80 gestures to convey 15 unique meanings as commands and questions (Bryne 2017). The meanings loosely correspond to the hypernym forms of various parts of speech categories [here, no, give, on, on?, play?], which are discussed further below.

Among the Hominins, stone tools provide the first evidence for advancement in behavior. The initial Mode I tools are currently dated to 3.3 Mya and remained at a relatively fixed level of design and refinement for over a million and a half years. The early hominins did not evidently undergo much generational change; contrary to the current human cliche “kids these days...”. Mode II tools were then developed and these spread slowly throughout the existing hominin range over the next million years. When humans first speciated, they inherited a Mode II toolkit. They had animal hides, cord, hafted hand-axes, spears, and cooked meals, to name a few of their initial conveniences. Hominins had been hunting elephants and hippopotamuses since at least 400 Kya and early human sites dated 200 Kya also contain evidence that they subsisted on these animals.

Humans then began making rapid advances to their toolkits (Henshilwood 2018) and then left Africa approximately 75 Kya. The sudden change to the rate of change of the toolkit just prior to leaving Africa is suggestive that, at the simplest, a single change could have taken place in the hominins to allow them to make these advances. Our current human language (faculty) is argued to be this change (Bolhuis 2014). The use of complex sentences would presumably have allowed the hominins to more easily accumulate knowledge and transfer it to each other and their children. Daily storage and recall of unique sequences would also permit hominins the ability to mentally reconstruct scenarios after they occurred, which would allow them to explain them to others and explore possible solutions when time permitted.

All available evidence indicates that the current human language faculty and cognitive functionality was completely formed before humans left Africa and that it hasn't changed since (Bolhuis 2014); which would be a signature of the single change in Africa hypothesis. The argument for this is that babies from any culture can grow up in any new culture and will readily acquire the new culture and language, which is taken to imply that no changes to the human language faculty or other cognitive functions have occurred in humans since we left Africa.

It is also worth noting that not all humans in Africa obtained the new toolkit. Human sites dated as recent as 30Kya have been found that do not show evidence of advancement beyond that of the inherited toolkit (Scerri 2021).

For the present inquiry, the two main historical developments of interest are the mammalian neocortex and the human toolkit change. The neocortex is viewed as having endowed mammals with the ability to conceptualize symbols, to form vocabularies of these symbols, and to input, store, recall, and utilize random sequences of these symbols. The human change is viewed as happening when the neocortex became large enough to support a new thought process and/or a new thought process was developed.

—

Sections

2 The Interface

We begin with one of the main minimalist assumptions of Linguistics, namely that communicators have some common internal neurological/symbolic representation of each word in their vocabulary. This is shown as the two blue “interface parcels” in the diagram in figure 1, which borrows from similar diagrams in (Pinker 1999 and Berwick 2013). In this example, while walking out of the woods, the two hikers roust a duck, which causes the duck symbol to activate in each hiker’s interface parcel. We assume that each hiker processes their unique neural input of the event and each conceptualizes, or activates, an internal symbol that corresponds to “duck”. Also note that if one of the hikers is replaced by another mammal, such as a dog, the interface parcel representation would still be valid.

For our initial purposes we assume that some analogous neural parcel exists that contains many neural attractor states that represent words or symbols. Such a neural parcel can be thought of as analogous to a two dimensional optical character recognition neural

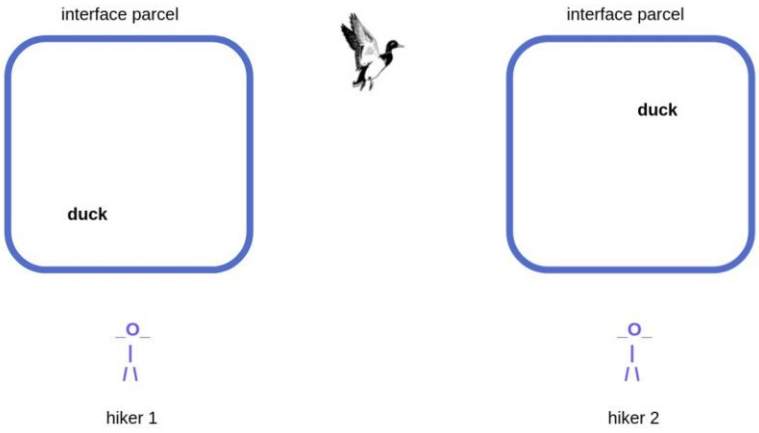


Figure 1: The rousting of the duck provides unique sensory input to each hiker's interface parcel, which we assume causes the "duck" symbol to fire (or conceptualize) therein. This is posited to occur as some subset of the nodes in the interface parcel firing into a stable attractor state.

network, wherein each character is a unique attractor state of the OCR neural network. Each character corresponds to a subset of nodes in the 2d array that fire and lock into the active state when the array is input a noisy bitmap of a scanned character and allowed to run freely into the nearest attractor state.

The symbols in this interface parcel, once formed, are assumed capable of subsequent re-activation on similar input. Furthermore, once activated, we assume that the symbols retain this state information for a short time and are more easily re-activated (Hubel 1980). This is analogous to the human ability to be given a random word and then be able to repeat that word on command some short time later.

The interface parcel can be abstracted to represent all of the symbols that we are physically able to internalize and externalize. The temporal sequence of all such symbols can be thought of as our stream of consciousness.

3 Short term memory, (stm) sequences and recall

Behaviors involving short term memory are easily observable in mammals and many other animals. In humans, we can easily form stm associations between any two randomly picked

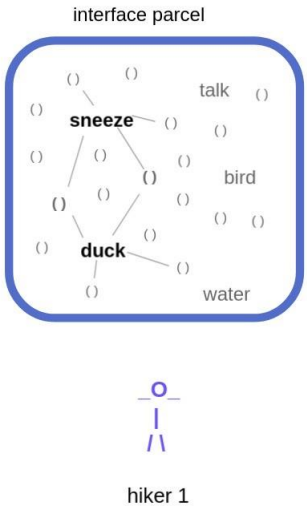


Figure 2: Many sparsely connected (stm) nodes are capable (under appropriate conditions) of forming short term association between any two symbols in the parcel.

symbols in our vocabularies.

As an example, we can extend the duck scenario above such that the surprise of the duck rousting causes the first hiker to sneeze. Following this, the second hiker's interface parcel would contain the activated symbols "duck" and "sneeze" and these would be stm bound such that if some short while later a second duck was rousted, then this would cause the second hiker to recall the sneeze symbol and to expect the first hiker to sneeze again. We can say that the second duck caused the hiker to "think of" the sneeze symbol.

Such a random two symbol stm mechanism can be built by the current model using the assumed node functionality. We (hypothetically) introduce many additional nodes to the interface parcel, referred to as stm nodes. These stm nodes are assumed to be equivalent to the symbol nodes except that they are unlabeled. The stm nodes are assumed to be randomly and sparsely connected to the labeled symbol nodes (Hawkins 2005).

When two random symbol nodes, such as duck and sneeze, are activated and fire, they each provide input activation to their respective stm nodes. If a subset of these stm nodes is common to both symbols, this subset can become activated over background due to having 2x more input activation than the stm nodes that are not common. The elevated input

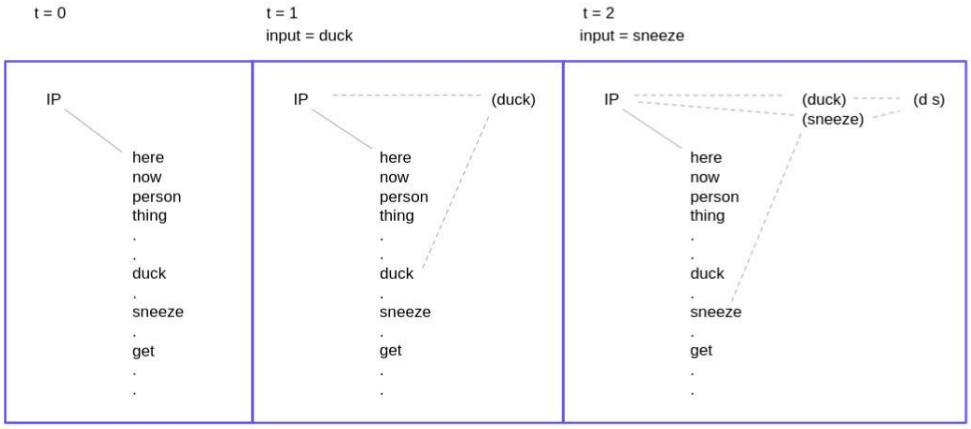


Figure 3: The child nodes fire based on external input and then touch and fire the ip head node. The ip node performs the merge between itself and the node that touched it. This creates an (stm) node as shown.

level is then assumed to persist for some time interval, allowing for subsequent short term associative recall.

Computationally, stm memory can be implemented as a single node created by a merge function between a root node “ip” (interface parcel) and its child nodes, as shown in figure 3. The (stm) nodes are created by the merge function that runs each time a node fires. Bidirectional connectivity is assumed possible in all connections.

The merge operation can be applied recursively between other recent stm nodes to create the (d s) node in figure 3, top right. This node then stores the short term association between duck and sneeze.

The stm nodes are created at run time using a merge constructor function of the Node class as shown in figure 4. This function takes two nodes as input, labeled “head” and “copy”. Bidirectional links are set up between the head and copy nodes in the merge constructor function.

The sequence: “hiker rocks duck sneeze” is input with the “touch” function calls in main() as shown in figures 5 and 6.

```
class  Node {

   string      symbol
   float       value                    //  [-1, 0, 1]  =  F,  ?,  T

   int         input_level              // current input value
   int         threshhold               // threshold to fire


          pair< Node*,  int weight >     head
          pair< Node*,  int weight >     copy
    list < pair< Node*,  int weight > >  branches


   Node()                         // base    constructor

   Node( string, head )           // string  constructor

   Node( head,   copy )           // merge   constructor


   touch  ( weight, depth )

   output ( tabs,   depth )

};
```

```
main() {

   Node   ip( “ip”, null   )

   Node   head( “head”, ip )

   Node   copy( “copy”, ip )

Node   child( head, copy )

};
```

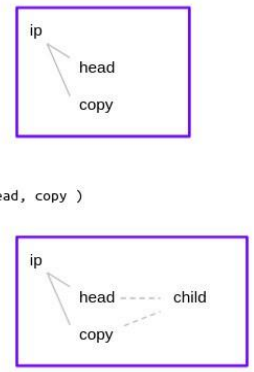


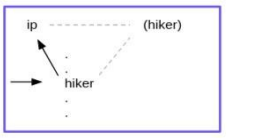
Figure 4: The Node class and the use of the constructor functions in main().

```
main() {
...

hiker  .touch( weight, null )

rocks  .touch( weight, null)

};
```



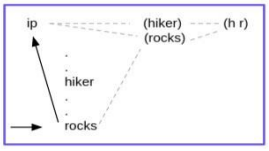


Figure 5: The hiker.touch() function call causes the hiker node to fire, which then causes the (head) ip node to fire. The stm merge node is created when the ip node fires.

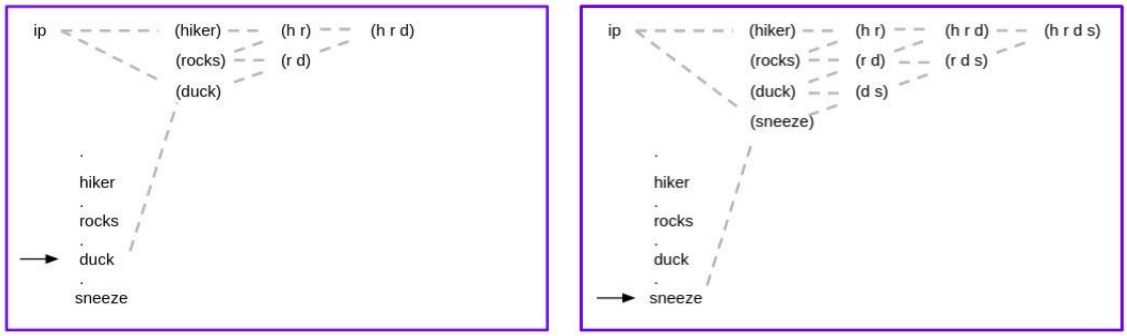


Figure 6: The stm structure grows iteratively as more symbols are input.

The structure formed in (stm) memory can then be used for output of the hiker, rock, duck, sneeze symbols. Using the recursive algorithm shown in the pseudo code in figure 7, the symbols can be output in the order that they occurred.

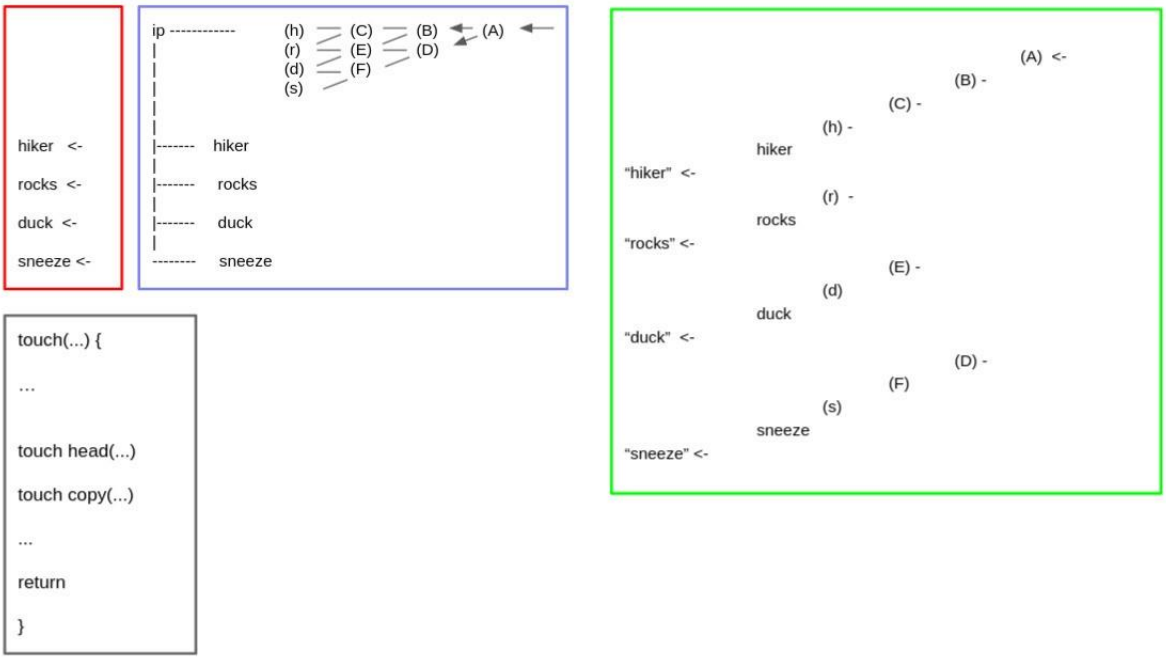


Figure 7: A c++ pointer to the rightmost (stm) node, denoted by "(A)", can be returned to main() and used to output the stored node sequence using the touch() function.

gesture	meaning	part of speech	hypernym form
grab, …	stop that	negative	stop
mouth stroke, … bite, …	acquire object contact (affection)	verb - get verb - feel -?	give touching
big loud scratch	init grooming	verb - point	do
arm swing, … beckon, …	move away move closer	verb - line - negative verb - line	go come
big loud scratch jump, … foot present, … reach	travel with me follow me (sex) climb on me climb on you?	prep prep prep ?	with with on on?
present location	groom here	adverb - here	here
	sexual attention:		
leaf clipping, punch ground leaf clipping	male female	? ?	flirt flirt

Figure 8: The chimpanzees exchange approximately 15 unique meanings using gestures. The hypernym forms are mapped to the closest meaning and can all be nominally matched. No grammar or random combinatoric use of gestures is observed. All species of Great Apes share a common set of 100 gestures with each species using a subset of 60 gestures, however, the gesture to meaning mapping is different in every species.

4 Protolanguage, Mammalian single symbol exchange

Following the suggestion of (Chomsky 2015), we assume a simple (truncated english) corpus of the form: [here now me thing get do go]. This corpus is drawn from the hypernym forms of the parts of speech: [adverb noun verb].

This initial corpus is similar to the reported meanings exchanged in chimpanzee gestural dialogues. The table in figure 8 is drawn from the meanings derived from a large video corpus of wild chimpanzee single symbol gestural exchanges (Hoabaiter 2014). The meanings are mapped to a part of speech and then to a hypernym word for that POS. The verb POS forms labeled ‘point’ and ‘line’ correspond to [do go].

Switching back to humans, our understanding of the neocortex is expanding at a great rate using many types of experimental methods. fMRI experiments in (Epstein 1998) and (Huth 2013) have identified two voxels ($< 2mm^3$) that fire in response to words that are hyponyms of (person/place) and (thing). Movies are shown to volunteers and the hypernym

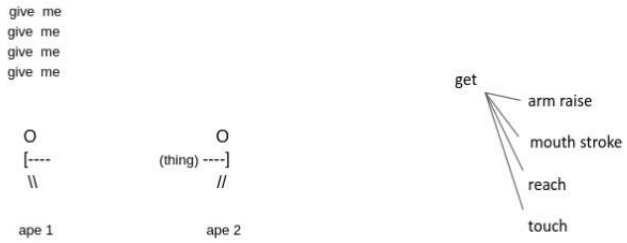


Figure 9: The apes use multiple gestures per meaning and young adult apes are sometimes observed to cycle through these as a repetitive sequence of gestures (Bryne 2017), which is suggestive of a hypernym-hyponym structure .

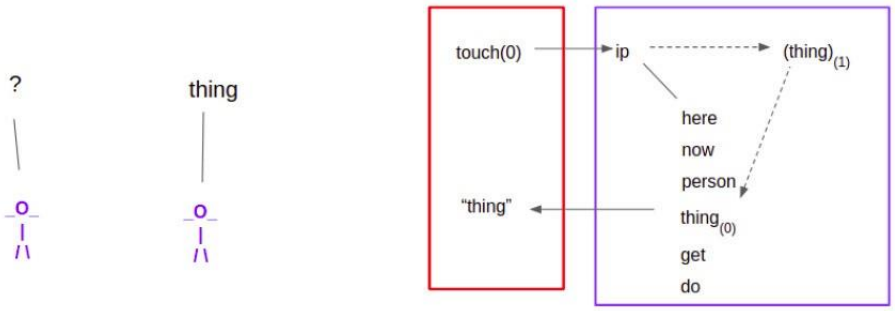


Figure 10: The value of zero passed in the touch(0) function indicates a question; -1, 0, 1 = [no, ?, yes]. The simplest input form of a question is the function call: ip.touch(0). The stm (thing) node is assumed to have been previously formed.

mappings from WordNet are used to tag 1800 nouns from the movie dialogues to person, place, or thing. In all volunteers, these two voxels can be identified in similar locations on a neocortex flatmap and show activation when the corresponding hyponym words are used in the movie dialogues.

The symbols of the proto-language are assumed to be formed as child nodes in the interface parcel as shown in figures 10-12. The stm memory allows for storage of state information and for simple dialogues.

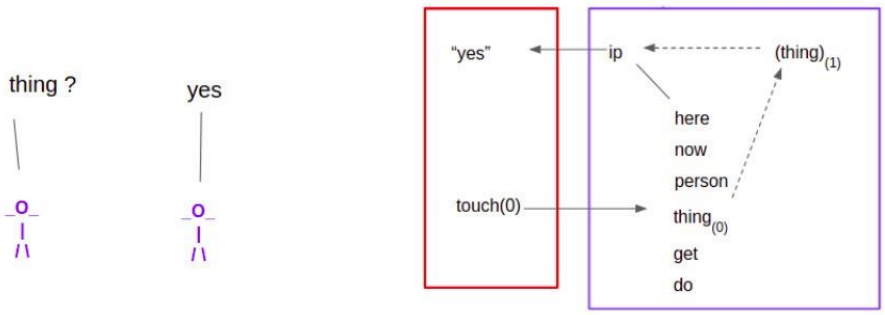


Figure 11: All symbols can be input as questions with the touch(0) function call as: thing.touch(0) .

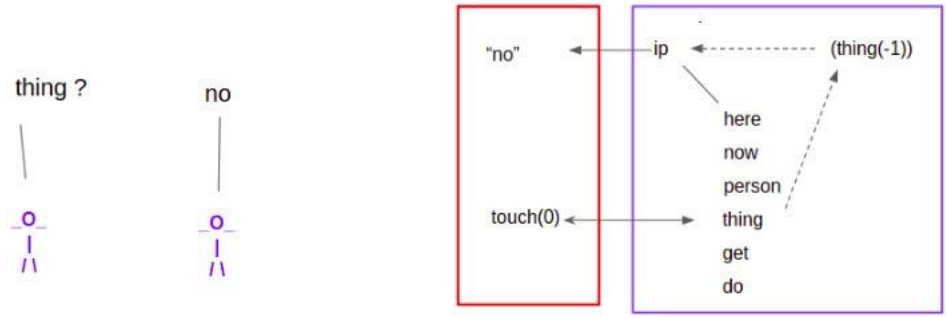


Figure 12: An stm node having a value = -1 indicates negation. In this scenario, (thing(-1)) was previously created via an stm merge.

5 Movement, boys eat what? what boys eat?

A mechanism for movement can be implemented by using the stored values in the (stm) nodes and modifying the pseudocode as shown. The input value of zero is propagated to the (stm) nodes as shown in figure 13.

This scenario is implemented in the c++ prototype as shown in figure 14. The blocks of text separated by horizontal dashed lines are static printouts of the ip graph (no arrows) or runtime graph flow diagrams (with arrows). These two outputs are made using the output() and touch() functions. The input sequence is "boys eat what", where what = thing:0. Following

input, a pointer the the (b e t) node is returned to main() and this is used in the call: (b e t)->touch() to invoke the output: “what boys eat”

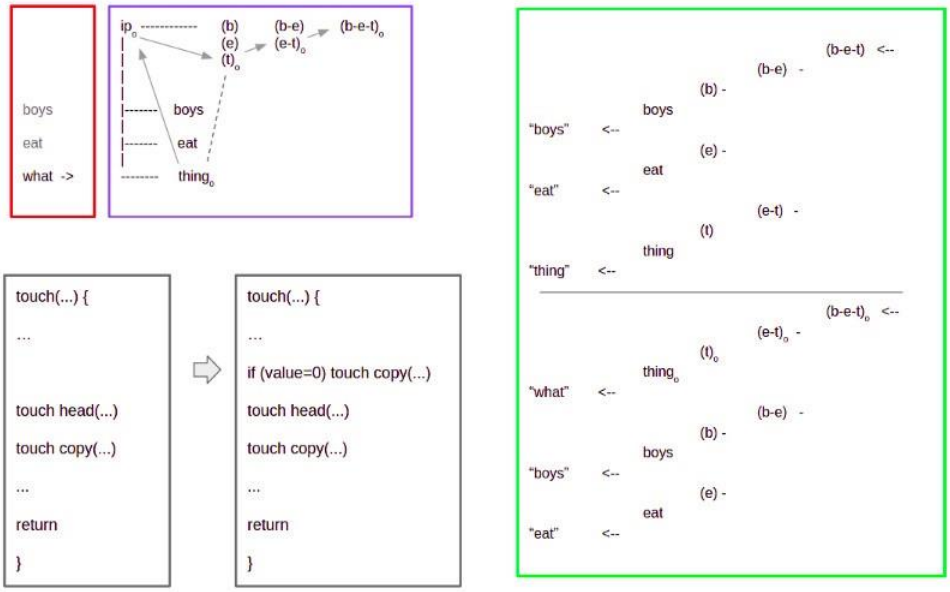


Figure 13: The single line modification to the pseudocode will alter the output ordering from “boys eat what” to “what boys eat”.

```
kwd1:code20$ c++ graph1.cpp
kwd1:code20$ ./a.out

------------------------------------------------------------

        IP

                        boys 0
                        thing 0
                        eat 0
------------------------------------------------------------

                        boys:1  <--
        IP:1  <--
        |
        (b)

                        eat:1  <--
        IP:1  <--
        |
        (e) (b e)

                        thing:0  <--
        IP:0  <--
        |
        (t) (e t) (b e t)

------------------------------------------------------------

        IP              (b) 1           (b e) 1         (b e t) 0
                        (e) 1           (e t) 0
                        (t) 0


                        boys 1
                        thing 0
                        eat 1
------------------------------------------------------------

 time interval occurs

                                                                (b e t):0  <--
                                                (e t):0  <--
                                (t):0  <--
                        thing:0  <--
                                                (b e):1  <--
                                (b):1  <--
                        boys:1   <--

                                (e):1  <--
                        eat:1    <--
```

Figure 14: c++ prototype output of “what boys eat?” Here “thing:0” is defined to be “what”

6 Adverb periodicity, here I now eat daily

The set of words in a language can be bifurcated and mapped into two symbols “a” or “n”, which correspond to the adverbs/adjectives and the nouns/verbs/prepositions.

The Wordnet English corpus of 120K definitions and 60K glossary sentences can be encoded into such reduced sequences and used as input as shown in figure 15. These

sequences can be considered a “truncated english”. The terminal nodes of the ip graph are then bifurcated further as shown in figure 16 to form a “less truncated english”. Oscillation of the a-n graph is posited to produce adverb periodicity in the spoken languages.

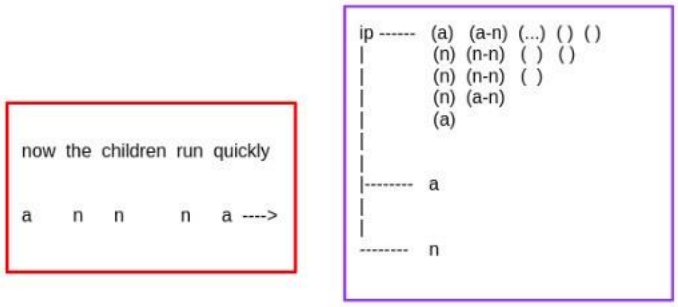


Figure 15: The 120K unique words in the WordNet corpus can be mapped to ‘a’ or ‘n’.

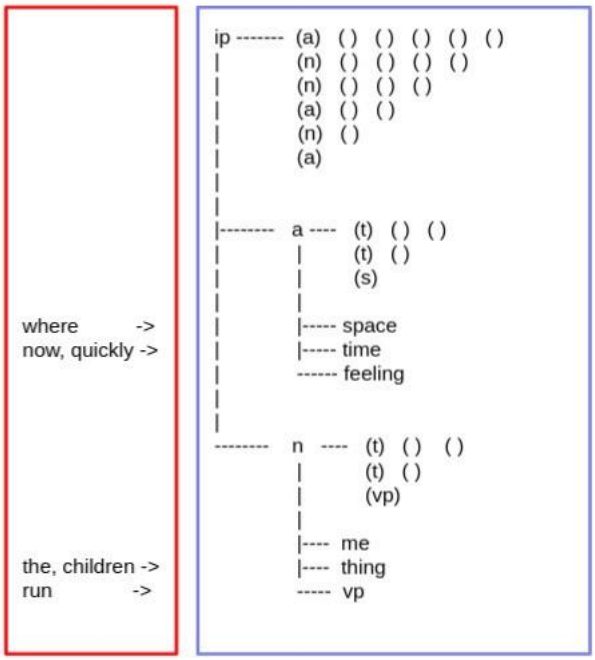


Figure 16: The a and n branches bifurcate further. The words in the corpus are mapped to one of the six hypernym forms. The stm memory mechanism functions within each branch of the graph. “now the children quickly run where?” is input and stored as “time thing time vp space?”

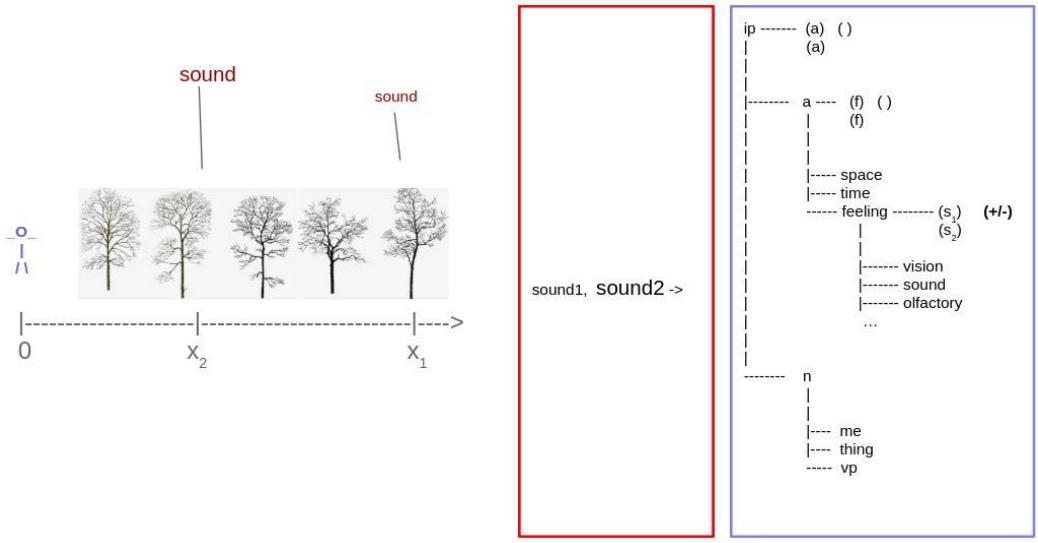


Figure 17: State information is stored in the adverb branch and can be used in a subsequent compare operation.

7 Compare function at the merge

here -here -HERE -HERE

A compare mechanism exists such that similar sensations, separated in time, can be compared. A familiar example is hearing the first two intermittent sounds of crunching leaves when someone or something is moving in the woods, relative to a fixed observer. The change in intensity is available at the interface parcel as an internal feedback to the observer.

The ability to compare two of the same sensory inputs separated in time is an important evolutionary advantage to all animals. The initial measured information from each sensation must be stored through time and then compared with the second measurement at a later time. This functionality can be achieved at the second level (stm) merge, shown in the adverb branch of figure 17, using the values of the two parent nodes.

This function, storing a value and using it in a compare operation later in time, is similar to that of the Reichardt model used by Hubel to explain directionally sensitive neural circuits in V1 (Hubel 1980).

8 Conjunctions, illicit conjunctions, movement of conjunctions

An “and” node is added to the IP graph, with the corresponding (stm) symbol labeled (+), as shown in figure 18. As introduced, this node would have no additional properties not already described.

An additional recursive touch() call is added to the touch() function to touch the (stm) node’s head as each node in an (stm) diagonal layer is added. This is shown by the arrows in Figure 19. In Figure 19, after “time” is input, the horizontal sequence of nodes that terminates in “||” forms a closed loop, which can be detected by the touch() function, allowing it to return an enhanced return value of 2 as indicated.

The return value of 2 can then be detected in the touch function and used to trigger a Hebbian enhancement of the weights between the calling and called nodes in (stm) memory.

An illicit conjunction sequence can be created with the model by introducing a single logical change to the (stm) nodes that are copy-rooted by the (+) node, namely, that if the return value in the Hebb loop is <2, then return -1. This is shown in Figure 20, where the (a +) node would detect the non closed loop condition (RETURN=1) and change the return value to -1.

The value=-1 is retained in the (stm) structure and can be propagated to subsequent (stm) recall. Such a mechanism could be used during sleep to exclude the stored (stm) sequence from normal [ltm] sequence storage.

Movement of a conjunction (when and where boys eat?) is possible by using the enhanced connection weight values stored in the closed loop. In the c++ prototype output shown in figure 21, the input sequence: “thing space:0 and time:0” (= boys where and when?) is input to the IP

graph. The (n a + a) node is then touched from main() to cause the output sequence corresponding to: “when and where boys?”

The conjunction node functions to create a layer of (stm) nodes that returns a false signal if a closed loop is not detected. Additionally, once a closed loop is detected, the subsequent (stm) nodes in the conjunction layer (example node: (n a +) in figure 21) can be disabled by storing a value=2 in the (stm) nodes. This process occurs locally, as each (stm) node is touched in the Hebbian loop that runs for each row of (stm) memory.

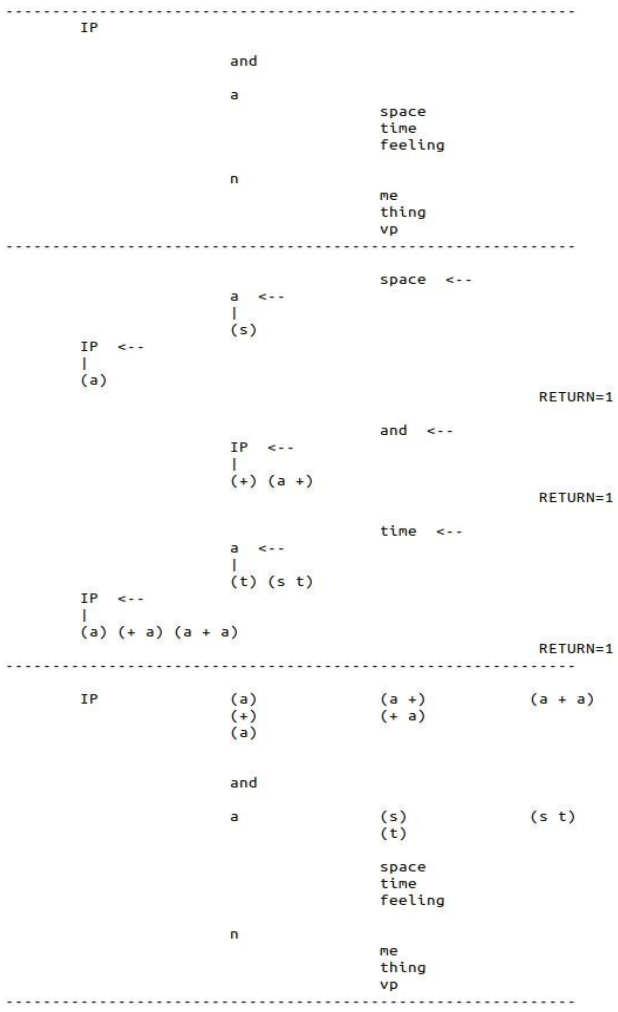


Figure 18: Inputting “space and time”.

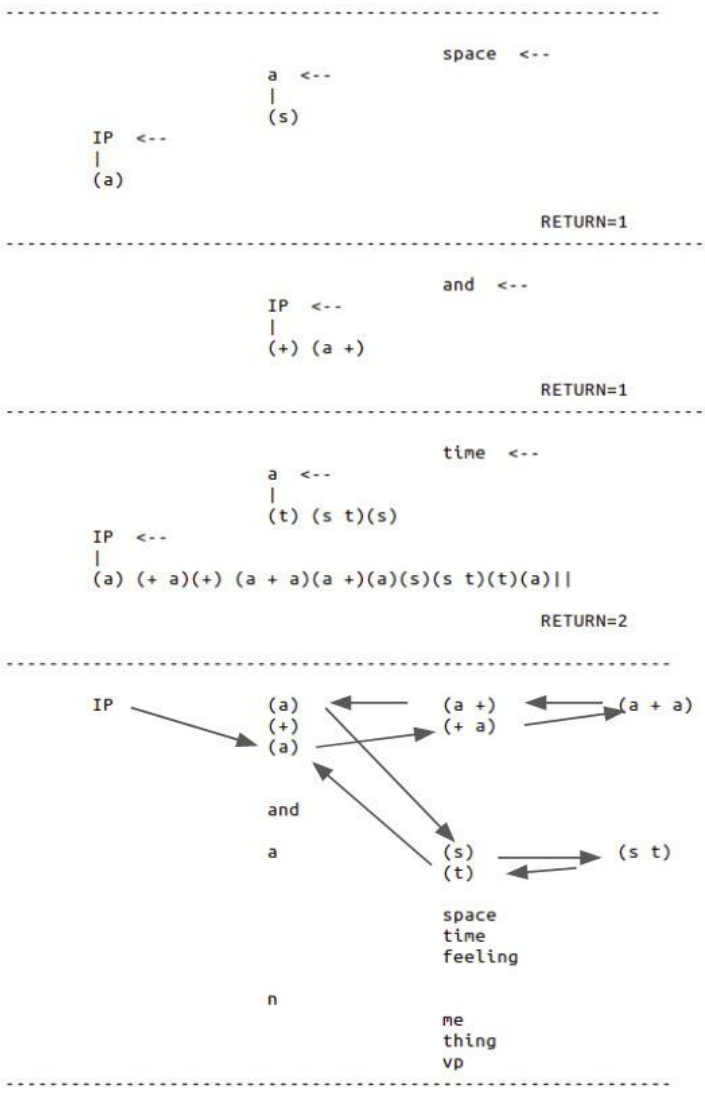


Figure 19: Closed loop detection and Hebbian one shot binding. The touch() function builds a list of node pointers as it recursively calls itself and can detect the closed loop and return an enhanced return value.

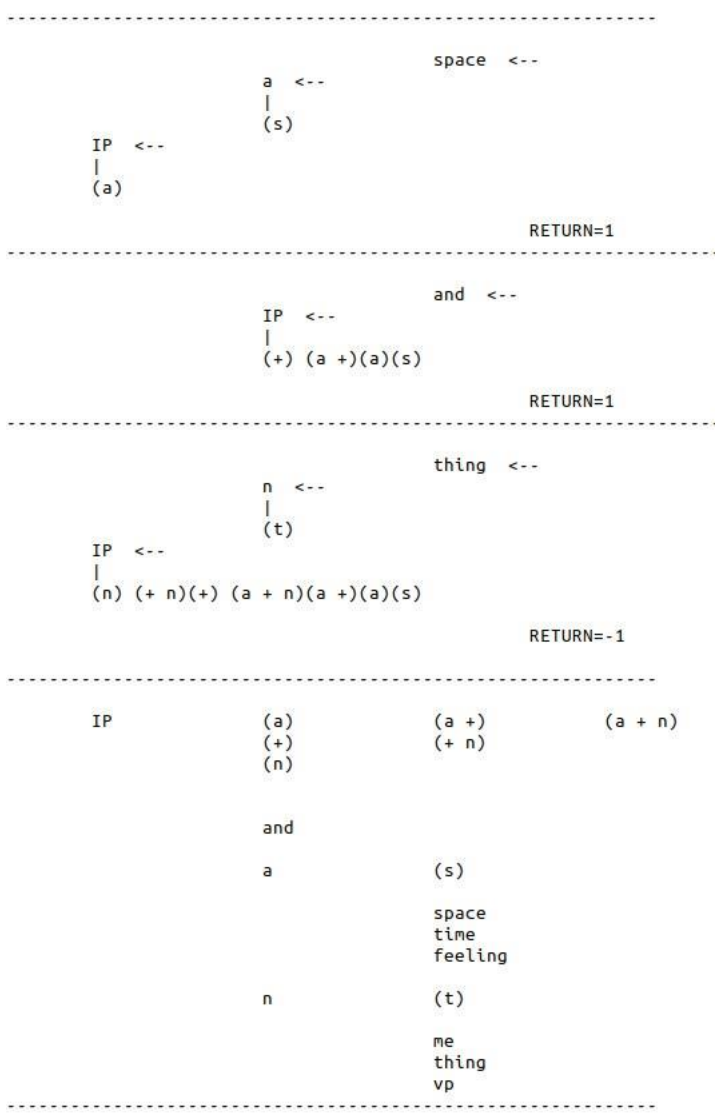


Figure 20: Illicit conjunction: space and *thing. A RETURN=-1 is generated by the (a +) node following the input of “thing”, based on the non closed loop condition (RETURN=1 from (a)).

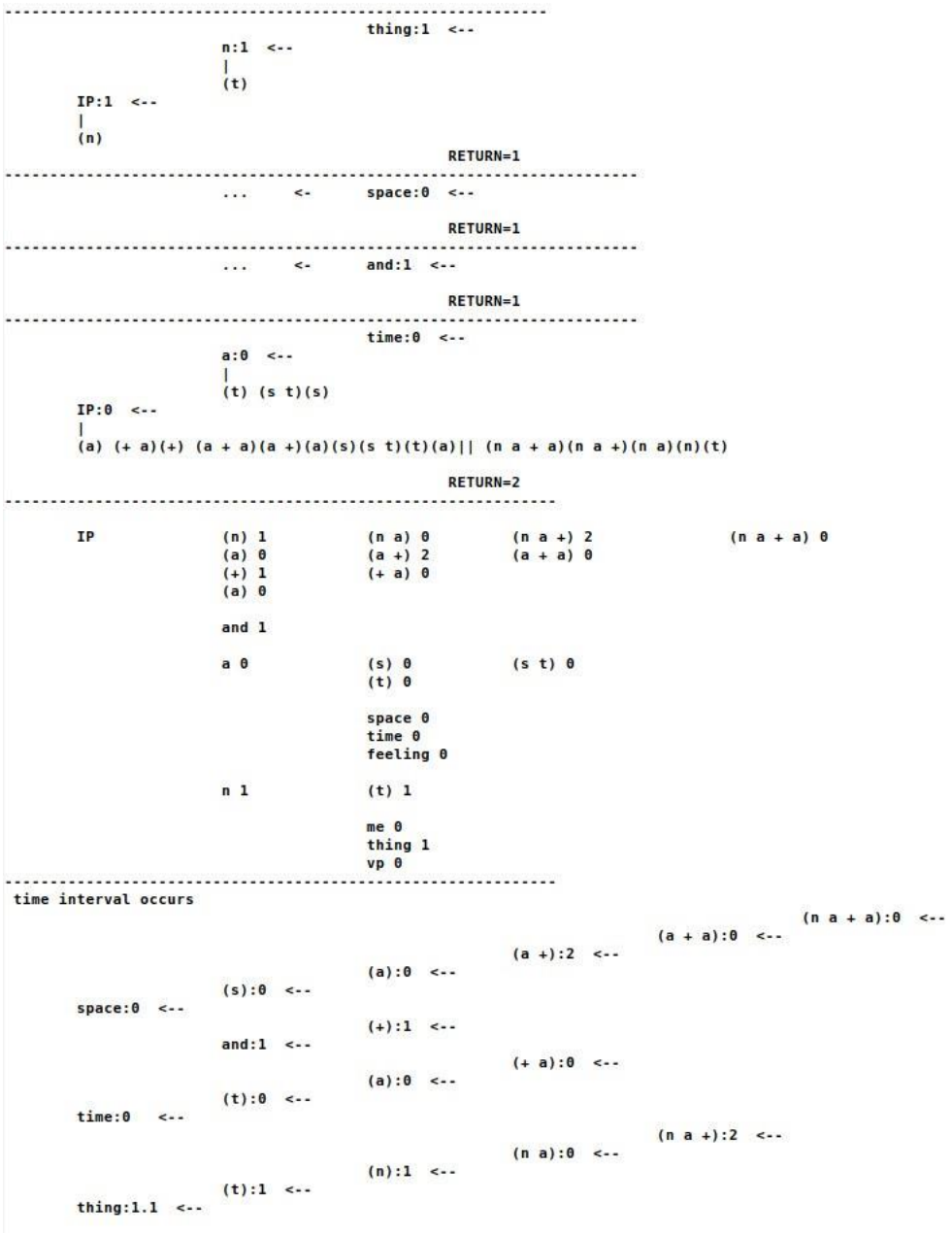


Figure 21: movement of a conjunction is accomplished using the modified bindings and the value of zero stored in IP (stm) memory.

9 Long term memory, [ltm] = merge(child, head), Sleep, Prediction

A permanent long term memory node can be made at the time the (stm) node is made by the merge() function, as shown in figure 22 using the square bracket notation: [ltm]. The (stm) and [ltm] nodes can also be connected by the merge function. The [ltm] node retains the label of the copy node as shown.

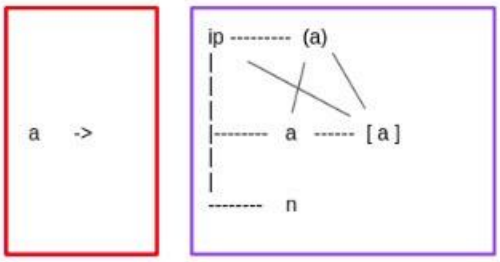


Figure 22: A long term memory node is formed by a reverse merge operation.

In the c++ prototype, the creation of [ltm] nodes by the merge() function can be toggled on and off with a single global variable. Once created, [ltm] nodes can be called by the (stm) nodes at the end of the existing touch() function using the link that was created at merge() time. This input sequence is shown in Figure 23. The call to the [ltm] node can also be toggled on and off; which would be analogous to inputting into (stm) rapidly without time to process the touch() call to the [ltm] node.

A dreaming mode is introduced in main() to replay the accumulated (stm) sequences and to build the associated sequences in [ltm] memory. This allows the [a n] node to be created as shown in figure 24.

Once [ltm] nodes have been created, the (stm) nodes can be cleared from the graph and

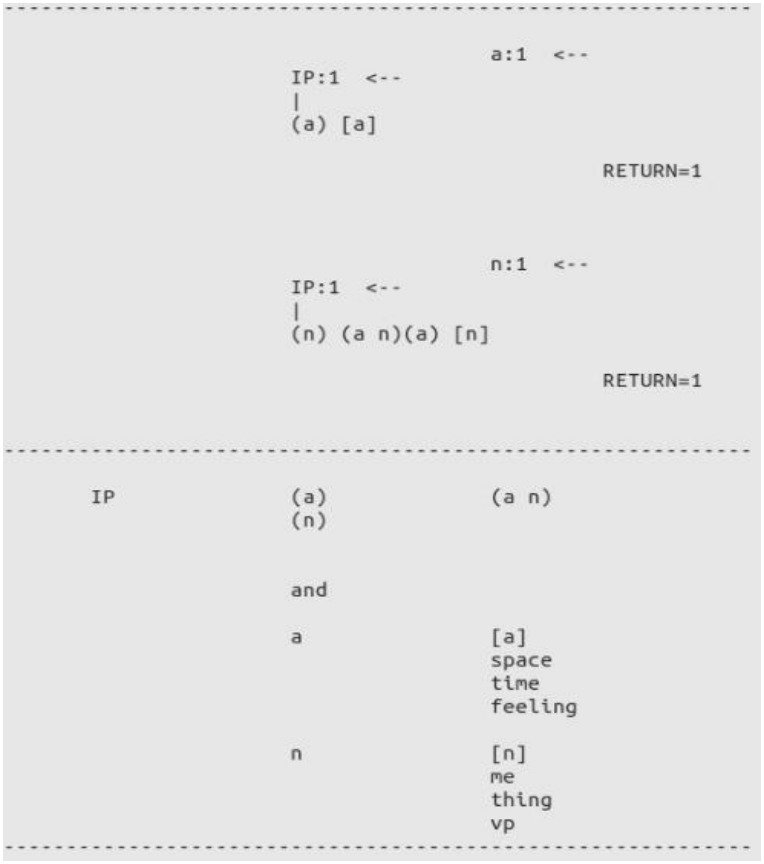


Figure 23: The [ltm] nodes fire at the end of the existing (stm) node sequences.

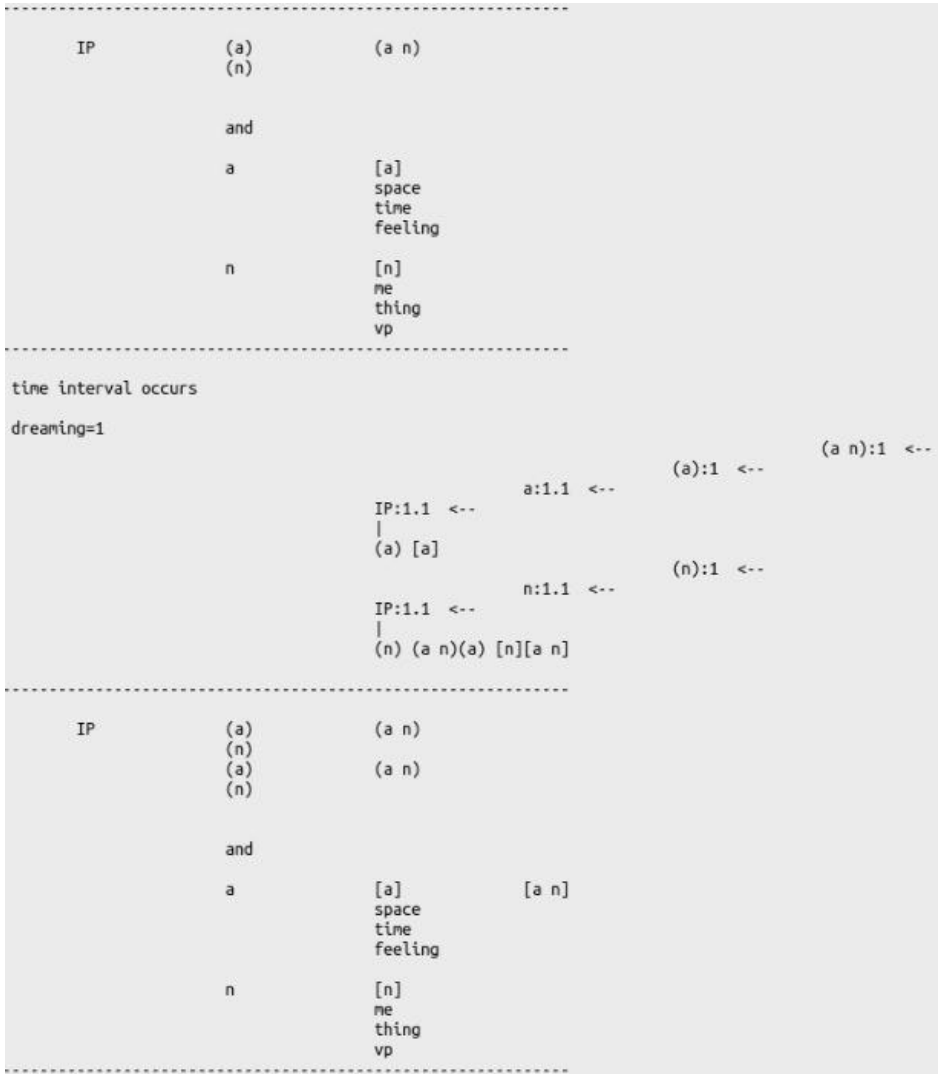


Figure 24: A dreaming mode allows previously created stm memory structures to duplicate in [ltm] memory by replaying the stored (stm) pattern with modified network parameters for the merge function.

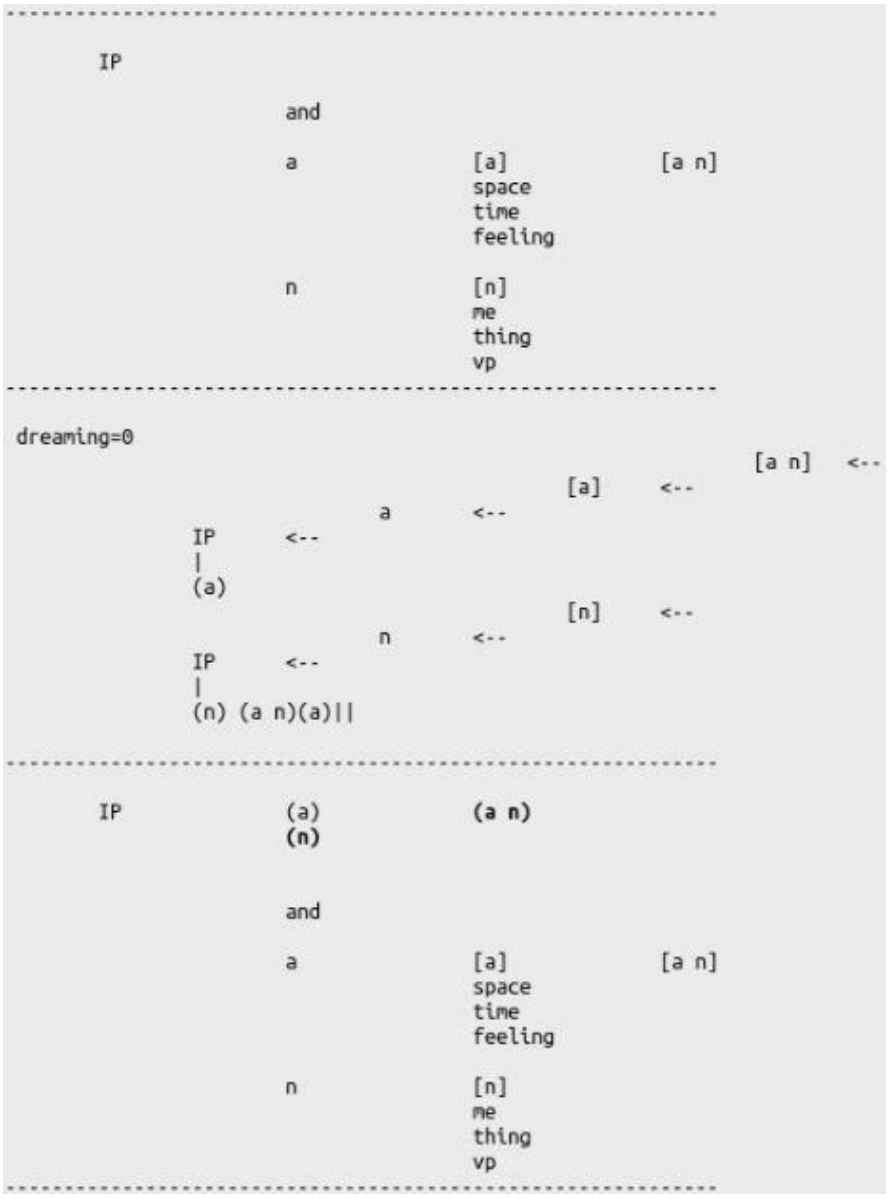


Figure 25: The [a n] node can then be used to recall the (a) (n) pattern to (stm) memory.

the [a n] node can be used to reproduce the node firing sequence that will rebuild the (stm) pattern in memory.

A simple prediction process can occur using the stored [a n] sequence, namely to predict an (n) node in (stm) memory following input of an "a" symbol. This is shown in the graph flow

shown in figure 26, where the symbol “space” is input, leading to the [a] -> [a n] -> [n] -> n -> IP -> (n).9 -> (a n).9 sequence to fire. Here the predicted (stm) nodes are created with value = .9 to differentiate them.

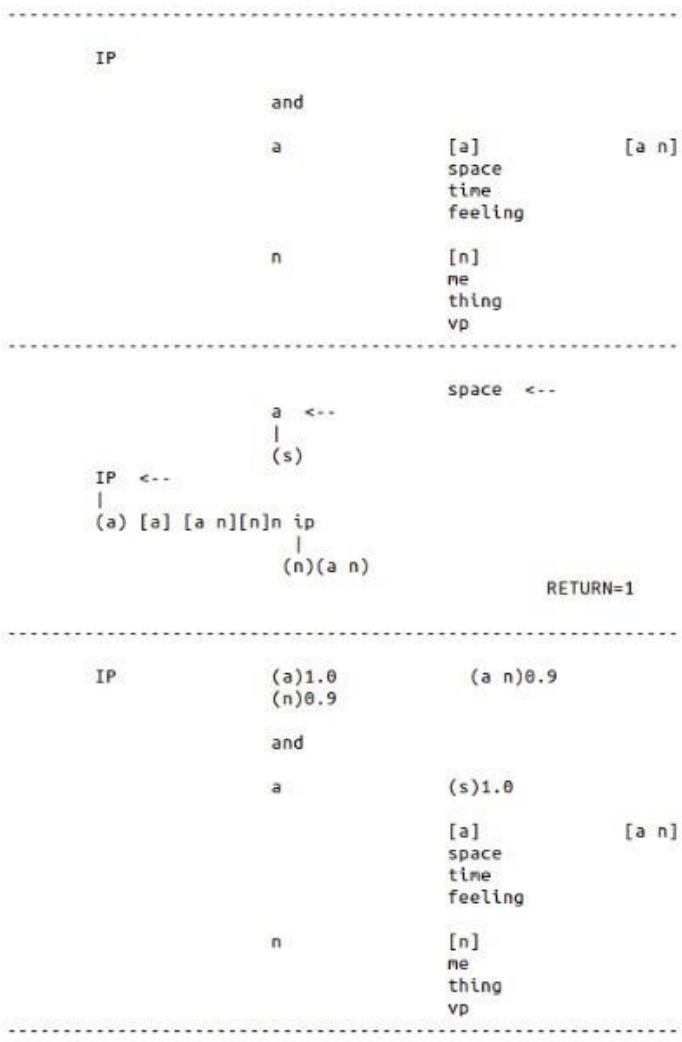


Figure 26: Prediction of “n” after the input of “space”

10 Verbs and prepositions

The head-connection weight of the [vp] node is manually modified in main() to cause its head node (the vp node) to fire when the [vp] node fires. This causes a second layer of (stm) nodes to be built in (stm) memory as shown in figure 27. This second (stm) layer can then alter subsequent graph operations, similarly to the "and" node.

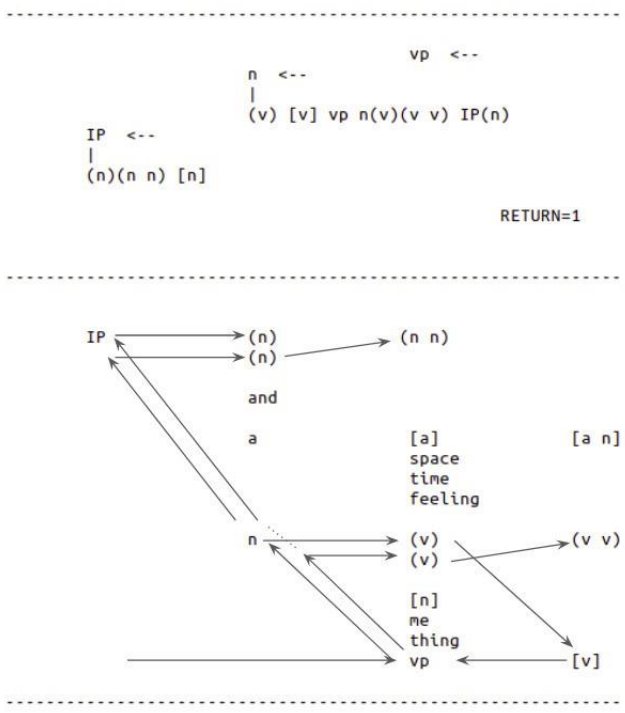


Figure 27: The vp node is the hypernym node for the verbs and prepositions. The weights of the [vp] node are such that it fires the vp node a second time, generating a second loop in (stm) memory. This second loop implements functions of the direct object.

In figure 28, "I eat food" is input, causing the (stm) layers to be built as shown. Input of "food" causes a closed loop to form (return=2) in nvp and ip (stm) memory. This would permit enhanced binding of the subject and object, using the same mechanism as the conjunction node.

We introduce a modification to the touch() function for the layer of (stm) nodes introduced by the second [vp] loop. This is shown in figure 28 following the input of "food"; a closed loop is detected in the nvp (stm) memory and a return value = 2 is returned to the calling nodes. This is posited to cause the $()_2$ (stm) node to touch() its copy node when the return value = 2 from the closed loop. This causes the graph flow to follow the (stm) memory back to the specific verb node and would allow for one shot Hebbian enhancement of the (stm) bindings between the verb and the subject+object.

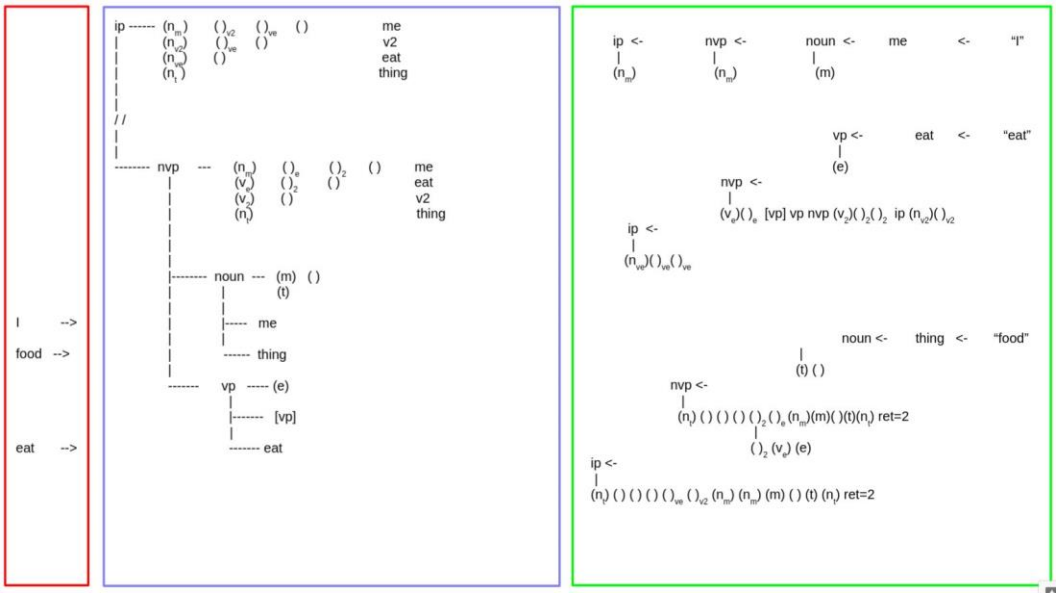


Figure 28: I eat food.

Illicit direct object constructions can be detected to cause a return value of -1. A purpose of this function would be to keep the illicit input node from forming enhanced bindings in (stm) memory by inhibiting the Hebbian one shot weight enhancement mechanism. This function can be implemented using the same mechanism as the conjunction node. In the graph flow shown upper right in figure 29, the input "I eat *drink" is depicted. Input of "drink" causes the $()_{v2}$ node to return -1 by the same rule as used in the conjunction $()_+$ node, namely if the return value (from the $()_{ve}$ node) is < 2, return -1.

This similar function between the $( )_{v2}$ and $( )_{+}$ nodes indicates that this function could be latent in all (stm) nodes and enabled by change of a local variable. This information would be stored in the permanent [vp] and "and" nodes and passed to the (stm) nodes at run time.

The second construction shown is "I eat now *food". Following input of the "now" symbol, the $( )_{v2}$ node would return -1 on the existing logic, however "I eat now" is a valid construction. This indicates the $( )_{at}$ node must override the return value = -1, as indicated in the graph flow. However, following the input of "food", the $( )_{at}$ node must then produce a return value of -1. This logic could be based on the condition of (return value = 2) + (the Hebbian enhancement already run).

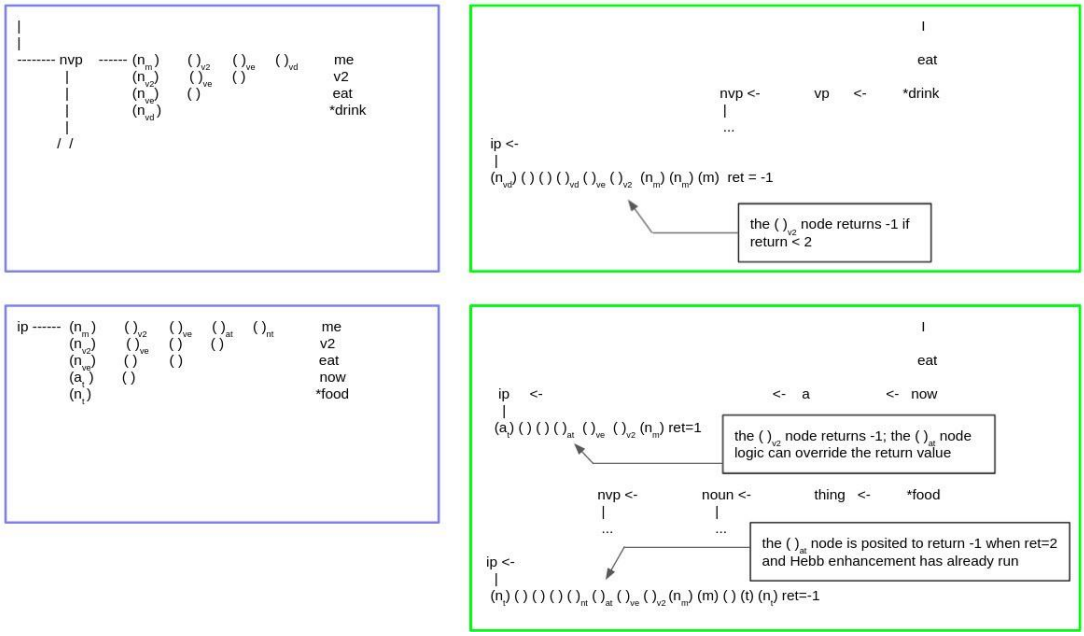


Figure 29: Illicit direct object constructions, "I eat *drink" and "I eat now *food".

Here the process of making functional changes to the touch() function is based on the desired language outcome. The recursive touch funtion is such that a single line of code change to the touch function will produced the desired changes to the graph flow.

Prepositions are posited to be children of a single node “prep” under the vp node. A manual merge is done from main() between the [p] node and the [s t] node to allow an additional layer of (stm) nodes to form in the ip node’s (stm) memory as shown in figure 30. This would allow the Hebbian one shot enhancement mechanism to run onto the “a” branch when “house” is input as shown.

Here we have combined space and time into a single node. This follows the pattern where the branches of the graph branch into two child branches.

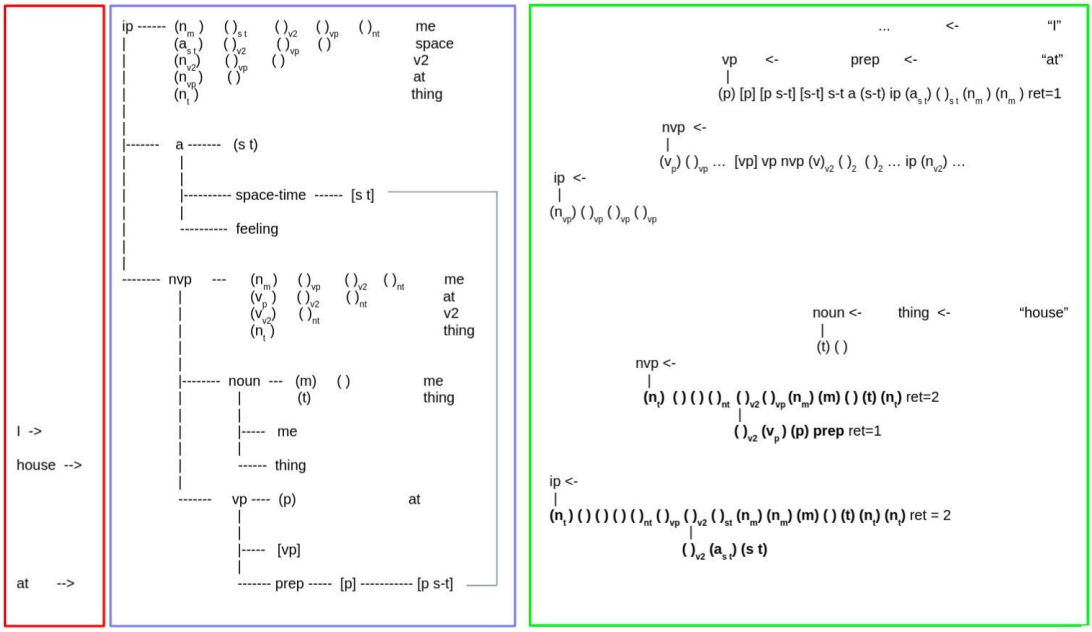


Figure 30: A “prep” node is created in main() with a merge([p], [s t]). The weights of the head and copy nodes are set so as to produce the (stm) layer in IP memory. Bolded nodes in the graph flow indicate the Hebbian enhanced weights following a return = 2.

Further bifurcations in the prep node would be: at, to, in, on... The pattern would be to create each new preposition node by a merge operation in main() between the preposition’s archetype and some other high level [ltm] node in the graph.

11 Past and future, progressive and perfected, singular and plural

The [ltm] nodes of the nvp, noun, and vp nodes are posited to develop pairs of child nodes, differentiated by a +-1 stored value, which correspond to the progressive/perfected tenses, singular/plural, and the past/future tenses, respectively.

This value based mechanism allows for illicit grammar detection in all three categories using the same mechanism, as described below.

11.1 Past and future, past irregular verbs

We modify the graph to add two child nodes to the [vp] node as shown in figure 31. These nodes are assumed to carry default values of +-1, corresponding to the past and future tenses. The "will" and "ed" tense markers are mapped to either node as shown. Upon input, the (stm) memory nodes are assumed to carry the +-1 value, which permits detection of the illicit construction when the merge function creates the (*) node in figure 31.

The $[-1]_{\text{PAST}}$ and $[+1]_{\text{FUTURE}}$ nodes are posited to cause an additional (stm) diagonal layer to form as shown.

The future modal verbs can be modeled in sequence, $[+.6]_{\text{can}}$, $[+.8]_{\text{shall}}$, $[+.9]_{\text{must}}$, $[+1.0]_{\text{will}}$.

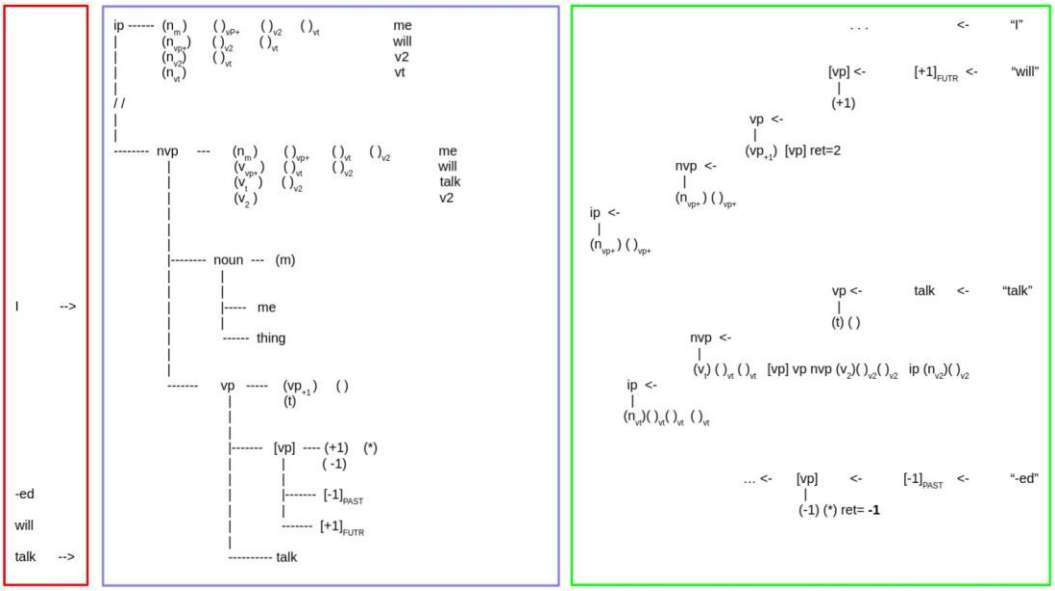


Figure 31: The vp frame node [vp] bifurcates on value to form the past and future tense symbols. Illicit tense detection (I will talk *ed) is possible at the merge of the (+1) and (-1) child nodes of the [vp] node.

Irregular verbs can be represented in the graph by a merge between a verb's [ltm] node and the $[-1]_{PAST}$ node. The graph flow at input is modified as shown in figure 32. The $[-1]_{ATE}$ node is assumed to produce the normal verb input but then to also follow its copy link to the $[-1]_{PAST}$ node, which produces an identical graph input response as the "-ed" verb ending.

This model implies that at some earlier time the irregular verb "ate" did not have an irregular form and that something caused the usage of the irregular form to be adopted.

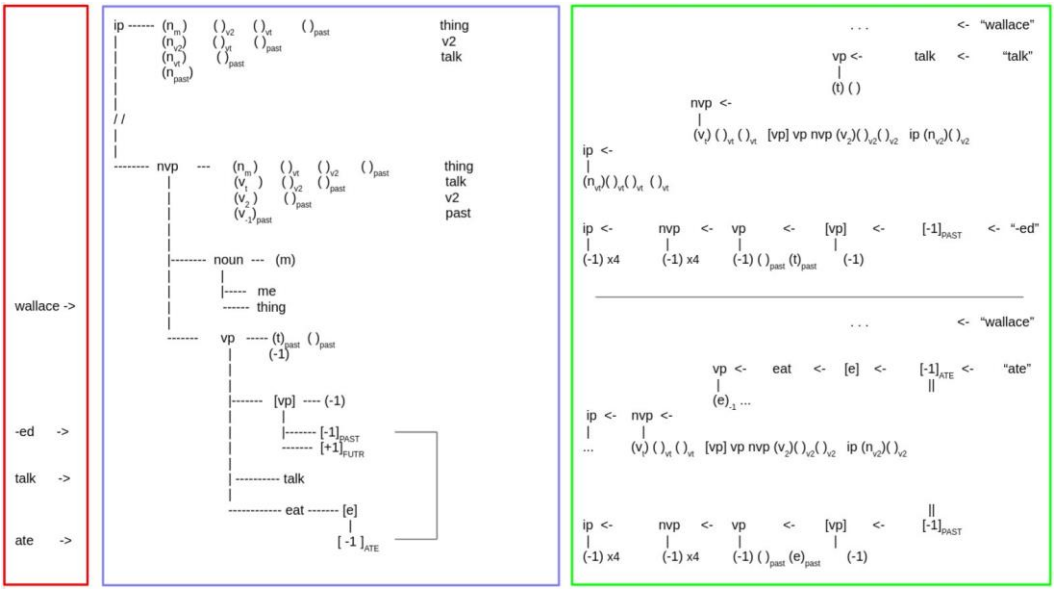


Figure 32: Irregular past tense verbs are formed by a merge between the verb's [ltm] node and the $[-1]_{PAST}$ node.

11.2 Progressive and perfected

The progressive and perfected tense symbols "are" and "have" are implemented on the graph using the same +-1 branching mechanism from the [nvp] node, which similarly permits detection of illicit tense grammar.

The $[-1]_{PROG}$ node is posited to merge with the [vp] node to create the [-ing] node. This permits a similar verb agreement check.

The $[-1]_{PROG}$ and $[+1]_{PERF}$ nodes can then form child nodes by merging with the $[1]_{PAST}$, [me], and [thing] nodes as shown.

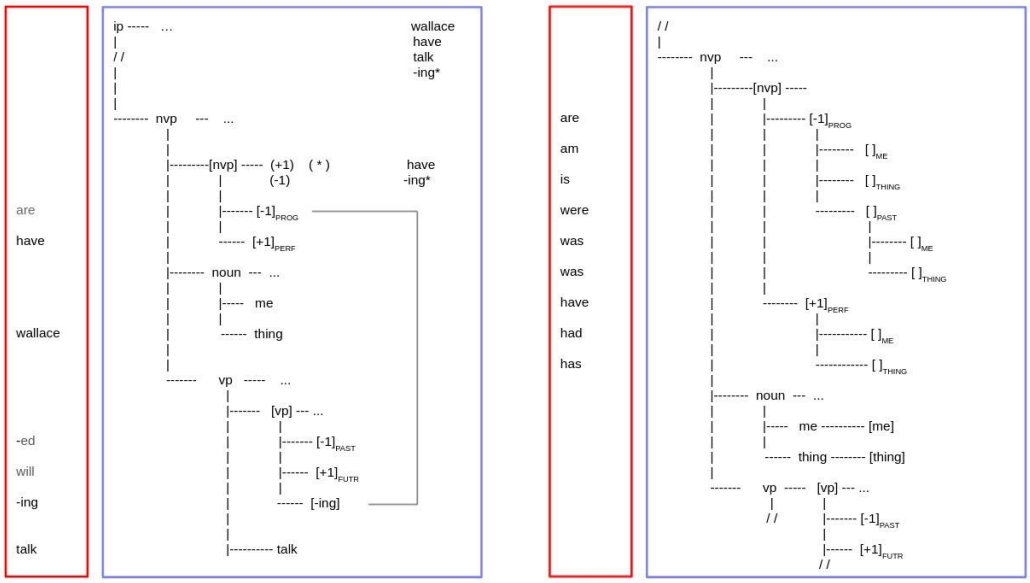


Figure 33: Progressive and perfected nodes develop as children of the [nvp] node. A negative signal can be generated by the merge in the [nvp] node's (stm), as in: wallace have talk ing*.

11.3 Singular/plural, my, the, a

Singular and Plural nodes are posited to form as value based branches of the [noun] node as shown in figure 34. "my" and "the" are posited to form as merge nodes between the [me]/[thing] nodes and the [noun] node. The [a] node is posited to form as a merge between the [the] node and the [+1]SINGULAR node.

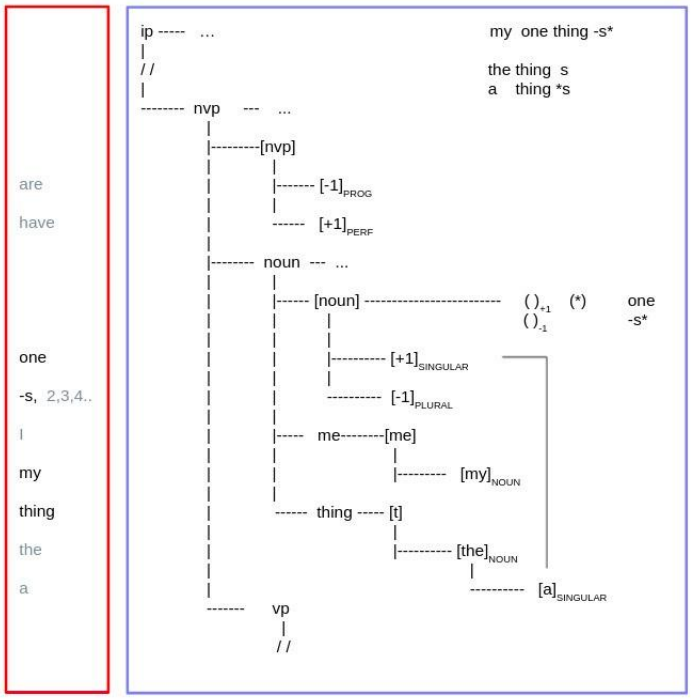


Figure 34: Posited merge nodes created for “the”, “a”, and “my” nodes.

12 Executive function in main(), input-predict-[respond], ponder-[respond], sleep

Input of a sentence sequence in main proceeds as a series of calls to the touch() function of each node in the sequence. The touch() function calls cause (stm) memory structures to accumulate in the ip graph, possibly forming enhanced Hebbian bindings, and then return a +-1, depending on whether any false signals were generated.

The (stm) memory structures that are built for each sentence can be probed from main() to return the rightmost (stm) node. These (stm) nodes can then be touch()-ed from main() to invoke sequence output.

Following an input sequence, the ip graph can be used to generate a prediction and/or a question based on its stored [ltm] and (stm) memory.

A ponder mode is posited whereby the same prediction mode would run, but with modified graph parameters to allow for deeper recursive function calls to probe [ltm] memory that could be recalled and utilized.

Repeating this process on a training corpus would then accumulate a set of (stm) nodes that can encode the sequences that have been processed since the last sleep cycle. A sleep cycle can then reprocess the (stm) memories into [ltm] structures.

The ip graph is then extended manually in main() by doing merge() function calls between existing terminal nodes, following which, the words in the training corpus are mapped to the new terminal graph nodes and the process of corpus processing is repeated, allowing for additional graph terminal nodes to be added.

13 Summary

A symbol is defined to carry a name and a signed value, with value being possibly zero or unspecified.

A symbol's value is analogous to the spiking rate of a neuron, with the sign being analogous to the relative phase of the spiking signal.

Symbols are assumed to be neurological attractor states formed in an "interface parcel".

All of our physical senses are assumed to be represented at the interface parcel as symbols that carry name and intensity (value).

The temporal sequence of all symbols that fire in the interface parcel, stimulated either internally or externally, is analogous to our stream of consciousness.

Computationally, the interface parcel symbols are implemented as hyponym (child) nodes of a single hypernym (head) node labeled "ip".

A short term memory model is proposed that operates via a merge operation between the hypernym and hyponym nodes in the ip graph.

The proposed stm merge operation can be applied to other recently created stm nodes to form short term associations between symbols that have recently fired.

A long term memory mechanism is proposed that operates via a reverse merge operation with the head and copy nodes swapped, but with the original copy node's label retained in the newly created [ltm] node.

Long term memory structures are formed in the same manner as (stm) structures. These ltm structures can be recalled to re-fire the symbols (in sequential order) in the interface parcel.

A single symbol dialogue system is assumed using a corpus of basic adverbs, nouns, verbs, and prepositions.

The ip-graph and short term memory mechanism allow for storage and recall of recently acquired information and for simple dialogue sequences to occur.

Questions and commands are implemented by calling the touch(value) function from main() with value = 0, 1.

The ip node takes on the externalizations [no, ?, yes] corresponding to ip.value = [-1, 0, 1] .

The (stm) memory mechanism allows any combination of two nodes in the ip graph to be associated together in short term memory.

A graph node labeled "a" is manually added to the ip graph. Symbols that are associated with a value, adverbs, adjectives, feelings, and sensations, map to the "a" branch of the ip graph.

A graph node labeled "n" is added to the ip graph which the remaining non adverb-like symbols map to.

At the simplest, a single oscillation of this graph can bind values stored in the a-branch to the last fired symbol in the n graph.

Oscillation of the a-n graph is hypothesized to produce adverb periodicity in spoken languages.

A diff() function can be done in the merge function at (stm) node creation time to generate a (greater than/less than) return value to the interface parcel (to main()).

A conjunction node, “and”, is added to the ip graph at the first level and operates in a-n (stm) memory as a new symbol type.

The (+) stm nodes created by the conjunction node are posited to cause a closed loop process to run on the graph which can detect a closed loop condition that can trigger a Hebbian one shot weight enhancement between the (stm) nodes in the graph.

The direct object node, labeled vp, is manually added to the noun branch and has its weights set to cause an extra ip-cycle to run that sets an additional diagonal layer of (stm) nodes. These (stm) nodes use the same closed loop detection method as the conjuntion node to implement the function of the direct object.

The verbs and prepositions will form as child nodes to this vp node and use the double ip-cycle mechanism.

A preposition node is posited to develop as a branch of the vp node that causes a second (stm) layer to form from the “a” branch.

The past and future tenses are posited to occur as a value based bifurcation of the [vp] node.

Irregular past tense verbs are formed with a merge between each verb’s [ltm] node and the $[-1]_{PAST}$ node.

The progressive and perfected nodes are posited to develop as similar value based bifurcation of the [nvp] node.

The current English grammar develops from these two nodes. The progressive node bifurcates on (merges with) past, me, and thing. Progressive-past bifurcates on me and thing and perfected bifurcates on me and thing.

Singular and Plural are posited to develop from the [noun] node using the same value based bifurcation mechanism.

The “my” and “the” nodes are posited to develop from the [me] and [thing] nodes through a merge with the [noun] node. The “a” node is positied to form through a merge of the [the] and $[+1]_{SINGULAR}$ nodes.

14 Acknowledgments

The author has received many helpful comments from many people who have reviewed early versions of this manuscript.

15 Ethical Approval

This article does not contain any studies with human participants or animals performed by any of the authors.

16 References

Berwick, et al, 2013, Evolution, brain, and the nature of language. In Trends Cogn Sci. 2013 Feb;17(2):89-98. doi: 10.1016/j.tics.2012.12.002. Epub 2013 Jan 9. See figure 1.

Bolhuis, Johan J. et al, 2014, How Could Language Have Evolved? In PLOS, August 26, 2014 http://dx.doi.org/10.1371/journal.pbio.1001934

Bryne, R.W. et al., 2017, Great ape gestures: intentional communication with a rich set of innate signals. In Anim Cogn DOI 10.1007/s10071-017-1096-4

Chomsky, Noam, 2015 Artificial Intelligence, in Navigating a Multispecies World: A Graduate Student Conference on the Species Turn. https://sts.hks.harvard.edu/events/workshops/navigatinga-multispecies-world/

Chomsky, Noam, 2015, Some Core Contested Concepts. In N. J Psycholinguist Res (2015) 44: 91.https://doi.org/10.1007/s10936-014-9331-5 https://dspace.mit.edu/openaccessdisseminate/1721.1/103525

Chomsky, Noam, 2016, "Language, Creativity, and the Limits of Understanding" (4-21-16). At https://www.youtube.com

Chomsky, Noam, 2011. Language and the Cognitive Science Revolution(s), lecture given at Carleton University 2011 https://chomsky.info/20110408/

Darling, Charles, "Guide to Grammar & Writing", http://grammar.ccc.commnet.edu/grammar/index.htm

Del Signore, 2014, Measuring and Simulating Cellular Switching System IP Traffic. In Bell Labs Tech. J., 18: 159–180. doi:10.1002/bltj.21651

Epstein R, Kanwisher, 1998 N. A cortical representation of the local visual environment. Nature. 1998;392(6676):598–601. The 'parahippocampal place area' (PPA) is described.

Fitch, 2017, Empirical approaches to the study of language evolution. In Psychon Bull Rev. 2017 Feb;24(1):3-33. doi: 10.3758/s13423-017-1236-5.

Hawkins, Jeff and Sandra Blakeslee., 2005, On Intelligence. Published by Times Books.

Henshilwood, C., et al, An abstract drawing from the 73,000-year-old levels at Blombos Cave, South Africa. In Nature 562, 115–118 (2018). https://doi.org/10.1038/s41586018-0514-3

Hobaiter, Catherine and Richard W. Byrne, 2014, The Meanings of Chimpanzee Gestures. In Current Biology Volume 24, Issue 14, 21 July 2014, Pages 1596-1600

Hubel, 1980, Eye Brain and Vision, David Hubel 1980. At http://hubel.med.harvard.edu/index.html

Hubel, D. H., 1957 Tungsten Microelectrode for Recording from Single Units. In Science 1957 Mar 22;125(3247):549-50. doi: 10.1126/science.125.3247.549.

Hubel, D. H., Wiesel, T. N., 1959 Receptive fields of single neurones in the cat's striate cortex. In J Physiol. 1959 Oct;148(3):574-91. doi: 10.1113/jphysiol.1959.sp006308.

Huth AG, et al, 2013, A Continuous Semantic Space Describes the Representation of Thousands of Object and Action Categories across the Human Brain. In Neuron. 2012 Dec 20; In 76(6): 1210–1224.doi: 10.1016/j.neuron.2012.10.014, Figure 2, PPA analysis.

Kruger N. et al., 2013. "Deep Hierarchies in the Primate Visual Cortex: What Can We Learn for Computer Vision?. In IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 35, no. 8, pp. 1847-1871, Aug. 2013, doi: 10.1109/TPAMI.2012.272.

Minsky, 1974, A Framework for Representing Knowledge, Marvin Minsky, MIT-AI Laboratory Memo 306, June, 1974. http://web.media.mit.edu/ minsky/papers/Frames/frames.html

Moore, Richard, 2014, Ape Gestures: Interpreting Chimpanzee and Bonobo Minds. In Cell Press. Volume 24, Issue 14, 21 July 2014, Pages R645-R647

Phillips, Collin, 2003, Linear Order and Constituency. In Linguistic Inquiry Vol. 34, No. 1 (Winter, 2003), pp. 37-90

Phillips, Collin, 2013, Derivational Order in Syntax: Evidence and Architectural Consequences. In Studies in Linguistics 6: 11-47.

Pinker, Steven, 1999, Words and Rules. At https://stevenpinker.com/files/pinker/files/edinburgh.pdf, figure 1.

Scerri, Eleanor M. L., et al., 2021, Continuity of the Middle Stone Age into the Holocene, In Scientific Reports volume 11, Article number: 70 (2021)

Tattersall I., 2016, A tentative framework for the acquisition of language and modern human cognition. In J Anthropol Sci. 2016 Jun 20;94:157-66. doi: 10.4436/JASS.94030. Epub 2015 Mar 25. PMID: 27014833.

repository of code used to generate figures: https://github.com/kwd2/graph1