On Making Modular Artificial Intelligence Systems

by

Daniel J. Barkalow

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

Master of Engineering in Computer Science

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

February 2001

Author . . . .
Department of Electrical Engineering and Computer Science
February 6, 2001

Certified by . . . .
Patrick Henry Winston
Professor
Thesis Supervisor

Accepted by . . . .
Arthur C. Smith
Chairman, Department Committee on Graduate Students

On Making Modular Artificial Intelligence Systems

by

Daniel J. Barkalow

Submitted to the Department of Electrical Engineering and Computer Science
on February 6, 2001, in partial fulfillment of the
requirements for the degree of
Master of Engineering in Computer Science

Abstract

In this thesis I discuss the need for multiple dissimilar parts in a system designed to show common sense. In the context of the reasons for this, I discuss the ways ithese modules must be able to interact and the ways they must be protected from one another. I propose an infrastructure which provides support for some of these requirements. In order to see what implementation details come up, I have implemented some parts of a natural language syntax system, which I describe. Finally, I suggest some ways my system can be extended and used.

Thesis Supervisor: Patrick Henry Winston
Title: Professor

Contents

List of Figures

Chapter 1

Introduction

The idea of an intelligent system extending beyond a single module is hardly new. Logic systems have always assumed input from some sort of perceptual system, and perceptual systems have assumed connections to something else, whether it be other perceptual systems or a central conceptual system. What has been missing from many projects is an explicit statement of this assumption, and a thorough examination of its implications. Systems consisting of a single module have been built that fail to provide a clear way for additional modules to be added, from the point of view of program structure and flow control. And systems have been built that fail to base their interfaces on an examination of what the systems on the other side will need and will be able to provide.

In this thesis, I look at the reasons that multiple dissimilar modules are required, and I explore the consequences for the system. This examination produces a set of requirements that an intelligent system must somehow satisfy, either in the infrastructure or in the way the modules work.

I then propose an architecture to support multiple modules of an intelligent system working together, taking into account the requirements I have found. This architecture does not violate any of the requirements, and provides infrastructural support for as many as it was practical to include. As much as possible, I have provided for future support for requirements not handled by the infrastructure or not yet identified. My architecture is by no means the only possible way to achieve the required behavior, but it is one arbitrarily chosen possibility, and some such choice must be made in order that a set of modules be able to work together.

Thirdly, I propose a module responsible for coordinating information exchange between other modules which are not directly attached. This module handles symbolic data which is shared between multiple systems. This module is responsible merely for presenting an interface between other modules in such a way that they can find the information they need from other modules and can avoid dealing with the information of types they do not understand. My proposal concerns the data which is shared between language and vision, and I base my choices of what information to exchange on features of those two fields.

As an example of the architecture and of the representation, I have written a system that takes textual input and generates conceptual structures. This has revealed some aspects of how modules written for this architecture should be structured and should work.

Finally, I speculate on what to do next. I suggest that a different method is needed for storing and searching the data my language model uses, and that this method may be used in other modules as well. I sketch some other modules that might connect to the conceptual end of my project and other ways in which the language module might interact with the conceptual module. And I suggest some global behavior patterns that might be used throughout the architecture.

Chapter 2

Theoretical Background

For many reasons, from the theoretical constraints to the history of artificial intelligence to the evidence from neuroscience, it is clear that any system which is sufficiently intelligent as to possess common sense (which seems to be the ability to apply pertinent knowledge to a problem) necessarily will be composed of multiple dissimilar parts.

The reasons this is true are important, as they point to features that a system must have in order to get benefits from modularity. These features will in turn have consequences for the types of architectures which are plausible foundations for comprehensive artificial intelligence systems.

The main points are: that information has to be broken up into manageable fields, such that the relationships between pieces of information do not overwhelm the pieces of data themselves; that the processing methods appropriate to different types of data are different and extension of a technique past the area it is good for leads to inefficiency; that strictly determined flow control does not allow the system to react appropriately to bottom-up pressures in determining processing; that any symbolic data the system uses must have sufficient information included to represent the entire meaning of the symbol, making it represent to the system everything it represents to humans; and that resolving ambiguity requires backwards communication.

2.1 Restricting the Size of the Knowledge Representation

There is a tremendous amount of information in the brain. Consider, for instance, the arrangements of things that you know: how your desk is organized, maps of buildings, maps of streets, and so forth. Or the list of names you know: famous people, brand names, product names, and place names, the names of people you have heard of who are not well-known, and the people you know personally [4]. Obviously, all of the data has to be encoded somehow. One significant feature of the data is that it contains a lot of possible connections and types of connections. Encoding all of the relations between data elements requires space per element which grows with the size of the data, rather than being constant like the data internal to an element.

Furthermore, the amount of relationships that may be represented grows faster than the information content of those relationships, as many relationships may be inferred, generally procedurally[1], from a much smaller quantity of information. If the system tries to lump all of this information together, it is likely to miss the procedural methods of determining information implicitly and store much more information than is necessary.

Even if the information is not stored, there are significantly more pieces of information which could be computed (possibly without sensible results) if all of the information in the system is mixed. The vast majority of possible relationships do not apply at all to the vast majority of pieces of information. If the system is looking for something blue, it must not only reject any ideas, fragments of speech, musical pieces, actions, and places it knows, it must not even take the time to consider them.

Thus the organization of the set of information as a whole must be such that it appears small from the point of view of any procedure which will search through it. If the addition of data not directly involved in a given relationship makes that

[1] Consider divisibility relations between numbers represented by binary strings. Symbolic inference is very difficult, but the relationships can be easily computed by a specialized procedure, and need not be stored.

relationship more time consuming to determine, the system cannot scale to the data sizes that the real world contains. This has been a common trap for systems which perform well in their areas of specialty; they expand the area of specialty, and thus involve too much information which could be categorically ignored if the system was more modular.

2.2 Restricting the Complexity of the Processes

Information required for intelligence comes in very different forms, such as visual input, models of objects, word meanings, accents and tones of voice, sounds, social relationships, and so forth. Different techniques are generally needed for each aspect, so a given project will be restricted to working on only one or a few at a time.

These constraints are sufficiently obvious that implementations follow them, even if the theory behind the project suggests that it should be more universal. However, solving a problem posed in one field with an answer in that field often involves information in a different field. In order to have a system that performs well, therefore, it must be able to handle information outside of its main field. Often, in order to make the performance of a system acceptable, it will grow in scope to include approximations of other fields.

For example, a language system could know that "one plus one" is "two", and a series of other relations. This would allow it to understand text with a certain amount of math in it, thereby making it work on a slightly wider range of sentences, reducing problems which use mathematical paraphrases to a more canonical form. However, the added complexity is large, as the language system will not have the right infrastructure to use the underlying math system to perform calculations; each equivalence has to be encoded individually. This example is, of course, a somewhat unlikely extension of a language system. But other, more useful, extensions, which are more likely to be included, are computationally even worse.

Approximating other fields in a project is highly inefficient, and generally makes it significantly harder to isolate the implementation of the main field itself so that the approximate section may be replaced with an accurate model. As the mass of approximations increases, the system becomes marginally more complete, but it does not scale well.

In order to have a complete set of efficient systems it is therefore necessary to have clear boundaries on what the system will do which follow the conceptual boundaries of the field. Initial systems made this way will not perform well, but they will be more able to connect to other systems which will handle the external information.

It is, in general, difficult to determine the point at which processing can be done better by a separate module rather than handled by an existing module which relates to the information. There is a tension between having the module fail to handle its field, either making other modules deal with too many details of the field or leaving inferrence patterns in the cracks between modules, and having the module be too extensive.

A module should handle as many features as can be treated exhaustively inside the module, as these will therefore not need to be exported to other modules, which would have to understand them needlessly. Of course, it should deal with and export any other features needed to handle the internal ones. It should not perform any further processing, since this is likely to introduce more complexity, as features and information which belong to other modules become necessary.

2.3 Coordinating the Processing

There cannot be a single thread of control between modules, since modules will, in general, pose questions and try to answer them internally, as well as allowing other modules to try to answer them. No part of the system can determine which module has the necessary point of view to solve the problem until some module actually solves it, and many parts of the system may not be able to determine themselves to be unable to solve a problem without a large amount of work. The modules must therefore work in parallel, either by taking turns explicitly or with multi-threading, and be able to stop processing if another system finds something.

For example, one module may try to search the memory for a certain date, while another module tries to figure out the answer from other evidence. The memory is large, and it may take an arbitrarily long time to search it effectively, if the information happens not to be easily accessible. If the reasoning module figures it out, the memory should stop trying to remember it, but otherwise, it will continue trying until it succeeds or loses interest. Neither approach should be used entirely first, as the system should be able to get quick access to any information easily available in either way.

Many modules can also provide the initial stimuli for a new problem. This requires each module which can do this to have some processing power at all times, looking at the input in case there is something important there. This allows it to present information to other modules when significant things happen, and also to make incremental revisions to its internal representations so that it does not have to analyze the data from scratch each time. This also lets it respond not only to the current information, but to significant changes in the information.

Since multiple threads of execution are clearly required, there is a question of whether the modules should take turns explicitly (either cooperative scheduling or periodic explicit calls into other modules) or should use preemptive multithreading and parallel processing.

Natural intelligence is clearly parallel in implementation, due to the nature of the brain. This does not indicate that it is explicitly parallel however; it might be such that it acts like a serial system. But that seems not to be the case; various sorts of behavior only make sense if they are effects of essentially timing issues. For example, people use irregular plurals and past tense words in many languages. However, even a person who usually uses the "correct" form ("correct" indicating here the form that the person would choose if asked explicitly) will on occasion chose the wrong one. This makes sense as a timing issue between coming up with the right word and speaking: the stem and regular ending may be found first, being more common, more primed, or just lucky, and the irregular form, which would be found eventually, is not found in time to replace the regular one.

Furthermore, there seems to be a data hazard; people will sometimes put the regular ending on the irregular form, creating a "double-plural". This makes sense if, at the output time, the irregular form has just been taken to replace the stem, but the regular ending has not yet been disposed of.

These effects, of course, are only a property of human intelligence, and not necessarily important for intelligence in general. But it seems to indicate that the benefits from having a multithreaded system, and, in particular, one without much synchronization, outweigh the chances for errors that it will cause. Presumably, the whole system is sufficiently fault-tollerant that these errors will not cause any long-term problems; the hearer will accept the "incorrect" plural, or an editor will correct it.

This serves to greatly simplify the interactions between modules. If each module has multiple threads, a module may ask another module for information and be answered without forcing the asked module to choose whether to work on its previous task or on answering the new question. New questions take processing time away from the existing tasks, but neither will be starved in reasonable conditions.

Programs are specified in three ways: data, control, and naming. Data is the actual information the program is working on. Control is the place in the process that the program is in working on the data. Naming is the way the program refers to the data. An interface has to specify the behavior of all of these aspects. The data is clearly necessary to specify, as that is the entire point of the system. Using multithreading and a relaxed thread model means that the control may be specified at a system level, rather than at a module level. This leaves naming, the way a module may refer to information from a different module and processes that module does. For the simple cases of modules which are tightly connected (such as phonology and syntax), this may be specified in the interfaces, but when the information is likely to be shared between a large number of modules which will be written at different times and may include modules which have not yet even been conceived of yet, it makes sense to specify some interfaces at a system level, such that there are conventions for the data format and the name scheme. I have done this for one such interface level, which I will present later.

2.4 Complete Meanings

One important reason that having multiple modules is necessary is that it permits the system to connect to the real world. Some theorists have assumed that meaning and, therefore, understanding, is universal and external to people. This is obviously not true, as the human brain is not directly connected to the outside world, and because many of the aspects which we ascribe to the outside world are simply not present in it. On the other hand, some theorists have assumed that the outside world is irrelevent to intelligence, and that meaning is purely internal. This is not true either, but for more subtle reasons: there are distinctions which appear to humans because of their peripheral perceptual aparatus, and which make no sense purely internally. There are enough of these, and they are sufficiently important that excluding them would make the system overly limited in scope, and, while their precise forms are probably not important, the experience of the real world as existing outside of the system as a source of information is a fundamental experience which connects people.

Consider, for example, colors. One could think of color as being an aspect of the outside world, and, for example, “green” as being universal. This is the attitude people are accustomed to adopting, and fits our experience best. However, it is simply false: consider, on one hand, a “yellow” piece of paper under a green light. It will reflects wavelengths of light in the “green” range, but, since our eyes adjust to lighting conditions, it will still be recognizably “yellow” and not “green”. Thus it is not a property of the light reaching our eyes. On the other hand, if a green spotlight falls on a white surface under otherwise white lighting conditions, we identify the color of the spot as “green” and not “white”. Thus it is not a property of the surface, either. It is only with the context of a significant amount of visual input, not coming from the object in question or the moment in question, that “green” is defined, and that context depends on the viewer.

On the other hand, there is little to no abstract significance to “green”. As a concept inside our mind, devoid of experience, there is very little substance to it: it is a color, and it is different from some colors (like “blue”) and related to other colors (like “emerald”), but the sum of these relations does not contain enough information. Consider the question of whether other people see the same colors that we do. If all of the colors in our mind were shifted around, the result would be entirely equivalent inside of us; talking to other people, we can tell that, if they saw “green” as “red”, but knew to call it “green” from experience, we would not know that they were different. Therefore, we at least believe there is something universal about colors, and we would certainly know if our perceptions of colors changed. Therefore, “green” is not entirely within the viewer.

Such concepts are based on the interaction of the outside world with the processing inside of the viewer. The same is true of objects in images, different recordings and arrangements of a piece of music, and so forth. The extent to which these are universal is the extent to which people in similar situations make the same judgements, having similar perceptual equipment.

In an artificial system, such concepts are almost certainly necessary, since they seem to comprise a significant portion of our concepts, and are much of the basis of the way we relate to one another. Furthermore, it is through these that we have the use of regularity in the physical world in being able to organize our internal representations.

If a system is to include a complete understanding of colors, for instance, it needs perceptual input to ground them. The same is true of many other concepts: verbs of manner of motion, objects classified by purpose or shape, sounds, and adjectives which describe these concepts, among many other examples. Many of these may be defined in relation to other concepts, but either still lack some aspect of their meaning, or the set of concepts as a whole lacks something, being defined circularly. In order to justify associating the internal representation in an artificial system with our internal concepts, it is necessary to represent this residual portion of the meanings, and the only way to represent it accurately is with a portion of the system which interacts with the physical world.

2.5 Information Flow

One important feature of human cognition is that information does not simply flow in a single direction. That is, information which must be computed later often appears to have effects in places that need to be handled earlier. Humans project information into the data which they infer from higher-level properties.

The simplest example is speech perception. In order to have any understanding of speech, it is necessary to identify the sequence of phonemes, as this is the common factor across different speakers and conditions. However, the phoneme sequence is not uniquely determined by the input signal; there is some variability in each phoneme. However, when people listen to speech, they hear the right phonemes, even when some of them are either too garbled in the signal to determine or left out entirely. And the phoneme sequence they come up with generally makes sense. However, it is not a matter of ignoring the phonology when higher levels don't like it; the result has to be consistent with what the person heard. Furthermore, the set of possibilities for a phoneme or a string of them considered as a word does not form anything remotely coherent at any level other than phonology.

One way of considering this is to say that higher systems get information in groups of alternatives, along with some sort of probability biasing mechanism. They then pass on information in such groups. The problem with this approach is that the system can never reduce the amount of data it is passing around, since any combination could turn out to be right, and has to be worked out. If every level both multiplies the possibilities because of all of the combinations at the lower level, and also introduces its own multiple interpretations for each possibility, the amount of data which turns out to be entirely mistaken quickly gets prohibitively big. And none of these possibilities may be weeded out, because any could turn out to be the only consistent result at some higher level.

Another way is to assume that the higher level system has some way of knowing how to reinterpret the lower-level data if it finds the original interpretation to be unsatisfactory. But this means that the higher level has to be able to understand the sort of data used at the lower level, clearly a duplication of effort.

A more likely approach is to have the lower level keep its data when it passes on its interpretations and be able to accept a higher-level complaint about a particular interpretation, possibly with some guidance. So the level which is picking out words from the phoneme sequence will send on its most likely interpretation of a segment, and the syntax system, having gotten a sequence of these which is probably mostly right, may tell it that certain words are not possible in context and perhaps say that the particular space requires a noun and not a verb. The word level might be able to find a replacement, or may have to ask the phoneme level for replacements for some phonemes; it might also change not only the word that the syntax system rejected, but adjacent words, if the problem seemed to be with the word boundaries; this could change what the syntax system wanted, and so forth. The word system could alternatively reply that it is sure of the analysis it gave, and that the syntax system will have to either come up with an interpretation that uses this analysis or give up.

If the input actually has a valid interpretation, this chain of interactions should find it, assuming that the analyses are reasonably accurate. It may find it more efficiently if each system keeps track of the possibilities it has tried for each position, so that, if asked to switch back to a former interpretation (if the third-most-likely possibility, for instance, has the same word boundaries as the first, but not the second). This is also most efficient in handling data where large structures are made up of smaller structures such that a change to one of the small structures normally doesn't affect the other small structures.

Language has this property: while a small change in a single word can change the meaning of the entire sentence, it is unlikely to, and it is very unlikely to change the meanings of other small parts of the sentence. Vision has it as well: if one object turns out to be different from the first analysis it was given, other objects in the scene can generally retain their interpretations.

Human language perception seems, from psychological and physiological data, to follow the model I suggest. People have the perception of having a single interpretation of a sentence at a time, and may switch to a different interpretation, but do not generally have the perception of having alternatives that they may choose between until they have switched between some. Furthermore, in reading, eye tracking data shows that, if a sentence turns out to have a surprising structure, the reader's eyes secade back to the point of ambiguity and review the sentence, rather than simply having had multiple interpretations generated in the first viewing.

Vision has a similar perceptual effect, shown by the Necker cube and other ambiguous images. The viewer will interpret the image as "flipping back and forth" between interpretations, rather than having multiple interpretations simultaneously available.

Chapter 3

Architecture

With the theoretical requirements in mind already, I will now propose an architecture which will meet those requirements. It will concern three levels: the thread system and its guarrentees, the ambiguity-handling library, and one central representation.

3.1 Objects

In order to give the system an idea of what the programmer considers a single piece of data, I made a simple object system, which does not include inheritance or type-checking. Each object should contain all of the information which is to be kept synchronized. The object system provides some methods automatically: a method to return the class of the object, accessors for the fields declared readable, and mutators for the fields declared writable.

Each object has a lock which must be held when a thread wishes to access any of the fields of the object. Only one lock may be held at a time, however, which prevents deadlocks; a thread releases any lock it currently holds when it needs another lock, and reacquires the lock it had before when it finishes with the other lock. This means that methods will have to have only brief critical regions (generally limited to the actual step of reading a value, which should return a value that the field had at some point, and not a partially-changed value). This means that threads can update an object atomically without any effort, although they may only change one object at a time. Higher level synchronization could be done by routing changes to a group of objects through a single object, thus protecting the group with a single lock. For almost all cases, however, the set of changes which must be made atomically is also local.

An additional mechanism is needed to allow a thread to determine that a different thread has already begun to compute a certain value and wait for it to return the result, rather than trying to compute the value itself. More generally, it is sometimes necessary to be able to have a thread wait for a result while not holding the object lock (so that the result may be computed). For this purpose, a set of functions are provided which will release and reacquire the lock at the correct time.

With the exception of the last set of functions, everything involved in locking is implicit. For most purposes, no stricter consistency than avoiding intermediate values is sufficient, and even when it is necessary to do an atomic action composed of multiple accesses, it is generally easy to avoid calling methods of other objects in the middle.

For example, if this were used for bank accounts, changing the value of the account by some quantity could be done as a method which did not release the lock while running, and even account transfers could be implemented by removing a certain amount from one account, and then calling the other account's method to add that quantity. The only race condition here is whether the money is put into an account soon enough to be used, and that depends on the timing of the transactions anyway.

3.2 Multithreading

This portion of my architecture is, unfortunately, rather underdetermined, because the work I have done is mostly within a single module (language) and thus I have not tested the various possibilities for interaction patterns, especially in timing and control flow modification.

If the timing issues and race conditions in intelligent systems are important, it is likely that there is significance not only to the fact that such issues exist, but to the approximate relative speeds of the various processes. Under this assumption, it is important to have a thread scheduler which allows the allocation of different sized timeslices to different threads. The issues of how these time allotments should be determined is obscure. There are two likely constraints: it must be able to use up all (or as much as possible) of the available processor time productively, and it must be able to maintain relative speeds of different processes.

Muliple levels of allocation are likely to be helpful. That way, processes which should proceed at speeds in proportion may be allocated proportions of the time from a pool whose total size depends on the level of system activity.

It is useful to be able to maintain per-thread information of various sorts, including pieces specified by individual modules and not the system as a whole.

Interthread communication and interruption is useful as well. Often a thread will turn out to be computing information that is not actually needed by anything, due to analyses elsewhere in the system. Either a better choice is found (such as an irregular plural before the regular plural is figured out), or the system has already used a previously determined value (such as when the system actually says a word, without having found an irregular form). In such cases, it is good if the thread that determines the calculation to be useless can stop the thread calculating it, rather than having the calculating thread check constantly to see if its result will still be needed.

It is also helpful to be able to pause a thread and restart it later. This lets the system switch back and forth between analyses without waiting for the processing to be in a completed state.

Ideally, threads would be associated with particular results which they are intended to compute, and they would be controlled by reference to their results. This would minimize the role of control flow in interactions, which in turn makes the implications of the data more explicit.

3.3 Information Flow

The way my system handles this requirement is as follows:

When a level passes on information, it does it in segments which may be questioned. In the handling of some information, the level produces all of the output possibilities it can, but then only sends on the one of them; when a suitable result is produced, if not suitable result has been previously found, it is sent to the next level, and computation of other results is reduced in priority. If a result is rejected, the computation of other possibilities is given priority again, and a new result is produced.

Since there is not a direct mapping between the outputs produced that may be questioned and the threads of computation which produce them (as a thread will likely produce several adjacent pieces of analysis), the higher level must be able to accept replacement structures that change multiple pieces but keep most of the pieces the same and handle them efficiently, not losing the work that went into the original construction if it is still useful. For example, if a noun is replaced with a different noun, the syntax of the sentence will probably not have to change significantly, at least not directly (the new noun might have different semantics, which could cause the semantic component to reject the structure).

It is important that the processing code be as stable as possible (in the sense of stable functions), since it is reasonably likely that parts will be changed slightly. In cases where the actual space changes, instability is unavoidable, but in other cases there might be instability if the code is done wrong: it might regenerate data structures which are the same, or, in particular, tell levels higher up that something has changed when the result is actually equivalent.

Chapter 4

Concepts

I propose a module which is used to transfer information between other modules which are not tightly associated. Various types of vision and various levels of language can probably communicate among themselves in formats that they have agreed on, but the information that language uses and the information that vision uses, even about the same objects, are significantly different. Furthermore, other modules will also work on the same data, and will want to know if the information they are getting from vision and the information from language are related. Therefore, it would be helpful to have a shared space which could contain nodes to connect information to, and a variety of information which will be comprehensible to particular subsets of the modules.

I suggest a representation for the data in this module. The representation itself is arbitrary, but such a representation must be chosen as a portion of the interface.

I suggest a set of features which are used by the language module and are possible to determine from visual data. These cover a substantial portion of the meanings of verbs of physical actions.

4.1 Representing Concepts, Categories, and Functions

Concepts may represent anything conceived of as having an identity: cups, a particular fork, Beethoven's Ninth Symphony, red, and so forth. A single concept may be represented with multiple objects, if multiple modules separately generated it. In this case, one of the objects will be chosen as primary, and information will be localized in it. Concepts contain pieces of information consisting of a function and possibly some arguments.

Categories are types of concepts, on the most abstract level. There is a category for physical objects, one for colors, one for events and states, one for places, and so forth. Categories are distinguished by how modules use the concepts involved. Concepts from two different categories will either by used for different purposes by some module, or will be used by different modules. Concepts will be written in the following format in this thesis:

$$\begin{bmatrix} \text{CATEGORY LAYER} \\ \text{FUNCTION (Argument, ...)} \\ \ldots \end{bmatrix}$$

There are four layers of concepts: tokens, which are individual concepts of the specified category; types, which are types of concepts at a lower level than categories; collections, which are groups of similar tokens; and materials, which are individual bodies of a continuous substance. These are exemplified by singular nouns ("the fork"), generic nouns ("forks"), plural nouns ("the forks"), and mass nouns ("water").

Types form a structure where subtypes are, in general, mutually exclusive specializations of direct supertypes. A type may have multiple supertypes, so long as they are independent. The fact that subtypes are mutually exclusive is entirely unsupported by the real world, where subtypes often overlap in some ways. However, it seems to be a fact about human thought that it has this basis. Thus humans will, for example, consider the platypus to be a mammal and not a reptile, despite the fact that it has some charactistics of each and could, if so desired, be considered to be a somewhat exceptional case of both.

Individuals are represented by a different set of concepts. These may have supertypes in the same way as types. In addition to being represented by concepts individually, they may be represented as collections (where the individuals may or may not be individually represented). Some types have continuous instances, which are somewhat more amorphous than the individuals conceived on as tokens; these individual instances are represented by materials. An example is water. Such concepts behave partially as tokens and partially as collections, and generally are from a different set of types than the discrete types, although the difference is not one of category boundaries, as materials and tokens of the same category often may be used interchangeably.

For some categories, there are only types, and no concepts of the other categories. For example, it is hard to imagine what would constitute a non-type [Color]: even the color of a single object has some variation, and the limiting case in narrowness does not really have the sort of identity needed for a concept. I assume that, for [Colors], all concepts are types, such that further specialization is always possible.

The question of how layer is represented exactly is thorny. A type does not determine whether its instances are tokens or collections, but it does determine whether they are one of those or materials. Materials contrast with both of the others sorts of instances, and share some features with each. Some types include both material and token/collection instances. I have chosen to specify layer simply as one of the four symbols, for simplicity. A more detailed look at the conceptual module will probably answer the question, but that is beyond my scope here.

The information in a concept is represented in terms of a function[1] which may have arguments. These arguments may be of some basic type, may be concepts, or may be some other sort of data, if the modules that use it share some data structures. The preference is for arguments to be concepts, although this is clearly not possible

[1]Mathematically, it is actually a relation, but I will call it a function for the sake of sharing terminology with the Jackendoff.

all the time, as there is information which this representation must connect to which will not be represented in terms of relations between concepts.

Functions may be defined as combinations of other functions or functions with some aspects preset. Thus a function AT-TOP-OF(x) may be defined as AT-EDGE-OF$([\text{UPWARDS}], x)$. This is simply a shorthand; it may or may not turn out to be useful in the actual system. I will use it for convenience in this thesis. Similarly, it will sometimes be useful to refer to a particular layer of a category, which may be given its own name; STATE is simply the material layer of the more general EVENT.

Information expressed by functions in concepts is information that the system believes, either because it experienced it or because it is definitionally true. Presuppositions have this status, although the system does not necessarily care much that they are true, since the concepts involved are not necessarily connected to concepts the system will use for other purposes. This is represented as follows:

$$\begin{bmatrix} \text{X} \\ \text{F}\,(...) \end{bmatrix}$$

The system can also use concepts generated by BE to consider information. This is what is does with information that has not been determined to be true, or that is being considered as a statement for some reason. Statements naturally have meanings of this form. This is represented as follows:

$$\begin{bmatrix} \text{STATE} \\ \text{BE}\left(\begin{bmatrix} \text{X} \end{bmatrix},\ \text{F},\ ...\right) \end{bmatrix}$$

The system considers a statement to be true if it represents the system's experience. This corresponds to:

$$\begin{bmatrix} \text{STATE} \\ \text{BE}\left(\begin{bmatrix} \text{X} \\ \text{F}\,(...) \end{bmatrix},\ \text{F},\ ...\right) \end{bmatrix}$$

If the information all comes from a single sentence, this is a tautology, but it more generally represents a situation where the statement states something the system believes.[2] It is worth noting that this system can represent and discuss paradoxes, as it can have concepts without being drawn into handling their consequences. Logic is just another module, not an internal aspect of the system, and so logical problems do not undermine the system.

Although most functions are conventions that different modules which share information through conceptual structures agree on, there are some which are meaningful to the conceptual structure itself.

The EQUIV function gives the canonical object for a concept. Objects with this function only exist to maintain connections from other objects, and references to such objects should be changed to the objects they point to in order that access to the actual information will be quicker and the redundant objects may be garbage-collected.

The INSTANCE-OF function gives a type to which the concept belongs. The concept by be of any layer; if it is a type itself, the type specified is a more general type.

The EXEMPLIFIED-BY function, which appears only on types, gives some concepts which are more specific than the type. These may either be individuals or types. If it is a list of more than one type, they are considered to be disjoint and exhaustive.

The AS-IN function provides a connection from an element of a compound concept to the compound. This allows important inverse connections between concepts to be expressed, even when the concept inside is deeply embedded.

4.2 Salient Objects

In order for multiple modules to communicate using the shared space of conceptual structures, it must be possible for one module to get references to objects created by other modules. Otherwise, the mere fact of having created an object in the shared

[2]Jackendoff originally took the two representations to be equivalent. In order to express the difference between the two situations, and to keep speculation from getting into the internal representations of concepts, I have distinguished them. Of course, his inference rules apply to statements the system has decided to believe.

data format will not be sufficient to actually share the data. For this purpose, the conceptual structure level of representation has a list of concepts which are considered salient.

For each module that interacts with the conceptual structure module, there is a salience list, which contains all of the objects which are salient and whose reason for being salient is relevant to the module. That way, if an object is of a category that the module has no relation to, or a query specifies a function which the module does not use, the module will not waste time considering it.

There are two uses for the salient objects list. The first is to let modules publicize new objects or objects which have just become relavent again. This allows, for example, a vision module which has just seen an event to correlate the information with a hearing module which has heard it. In general, modules will look at newly publicized objects to try to determine if any of the objects that they have created might represent the same thing.

On the other hand, a module might want to use a property of an object which it does not compute and which has not been computed. In this case, it makes the object salient with a query matching the property it wants to know. Modules which compute this property then may try to compute it and fill it in.

Objects stay on lists until removed. In general, they are handled in order by arrival time, although that is up to the module to decide. In general, a module will remove an item once it has finished working on it. The module which put an item on the lists can also remove it (particularly in the case of queries, where other modules may still be trying to answer the query even after the initial module is satisfied with an answer).

This is essentially the simplest possible mechanism for coordination. Each module works on whatever it finds interesting, and is not constrained in when it produces results. A module which is trying to get information has the option of accepting any result it is given or even of dropping the query. Modules which will have no possibility of having information on a query never even hear about it, and continue working on other things. Modules may talk to themselves through the salient objects mechanism, and thereby intersperse computations for their own purposes with computations to resolve queries from other modules.

4.3 Concepts for the Physical World

The main category of physical world concepts is THINGS, which represents physical objects of all sorts.

PLACES are generally based on a reference object and represent position and a few other properties. These are two kinds of PLACES: non-distributive ones, which indicate that a THING occupying them is at some location fitting the conditions and distributive ones, which indicate that a THING is at all locations fitting the conditions; the latter case generally requires a plural or material THING.

A PLACE can also indicate that a THING occupying it is attached to, supporting, or in contact with the reference object; contact is implied by each of the other two, but attachment and support are independent. A PLACE that specifies these properties may or may not also specify location.

[3] mentions contact and attachment as features of prepositions. The meaning of "on" seems to rely on a feature of support, and leave the location features of the place function undetermined, with a preference for "above".

(1) a. I put the clock on the table.

b. I put the clock on the wall.

c. I put the gum on the bottom of the table.

In the first example, the table supports the clock by virtue of being underneath it and in contact; no attachment is implied. In the other two, the supporting objects are not underneath the other objects, but still support the other objects, now by virtue of attachment. This suggests that the important characteristic for "on" is support, and it is a matter of the physical model how that is achieved. This sort of data could be described by a preference model of the sort Jackendoff invokes for "see" (preference for vision, preference for attention, requirement for at least one), but in the case of "on", there seems to be a preference against attachment, even when it is plausible (the clock being glued to the table in 1a).

Furthermore, the connection between support and "on" explains the use of "on" in constructions such as "rely on" and "depend on", where "support" is also appropriate. If we intend to consider these uses as application of a physical world model to a non-physical domain, we need to account for the fact than "on" is widely used, but "atop" and other prepositions describing locations are anomalous.

(2) I based my case on/*atop/*above this data.

This indicates that Places may include information about support, contact, and attachment in addition to, and sometimes in place of, location. A Place is thus, despite the name, a description of a possible physical status for an object.

A Path is conceptually a sequence of Places. It may specify the start, the end, and points in between. It may specify that the Path does or does not reach either of the end points.

The world model includes several functions. Most prepositions correspond to functions which give Places relative to a reference object which is a Thing. Paths are generated by Toward, Away-From, and Via, and have additional features for determining if the endpoints are reached. Occupy produces a State combining a Thing and a Place. Extend produces a State combining a suitable Thing and a Path. Go produces an Event from a Thing and a Path.

4.4 Grounding the Concepts

Symbol systems, therefore, have to have their symbols grounded in perceptual systems. Based on this premise, it is important to know whether the symbols in my system correspond to anything perceptually accessible.

If this representation is to bridge the gap between the perceived physical world and language, the information it encodes needs to be sufficient to make all of the distinctions relevant to language. It also needs to be possible to generate perceptually.

Jeff Siskind has shown [8] that a simple vision system with certain ideas about the physical world can determine contact, attachment, and support relationships, and that such information is sufficient to accurately classify the sorts of events that involve motion of objects.

So support seems to be a useful addition to the system involving contact and attachment, and Siskind's work seems to validate the use of all three features, as a perceptual system can recognize them when they occur and make inferences based on them, as it has to make inferences to determine them in the first place, as none of them are actually explicitly visible, but much be inferred from the failure of objects to move in certain ways.

This final fact gives a further validation of my method: the inference patterns for support are seemingly simple but notoriously tricky. The traditional approach has been to attempt the problem inside the symbol system, which involves many different cases. But that portion of the logic is entirely unnecessary, as it would simply duplicate the work of a perceptual system which is designed specifically for this task.

Chapter 5

Language

To determine whether the architecture I designed is actually usable and the conceptual structures I have proposed are plausible, I wrote a module which uses its features to parse strings into these structures. I decided to base my methods as much as possible on current linguistic theories, since I wanted to have a chance of covering the full range of English sentences. My goal is to have a system able to read text intended for humans, and so I wanted the system to work on the same principles that human language capabilities work on.

I have based my theory on the work of several linguists and cognitive scientists, notably Ray Jackendoff, David Pesetsky, and Steven Pinker. Certainly there are innumerable others whose work I have used, but whose ideas have been more integrated into the canon. In particular, I have used the division of labor in morphology suggested by [4] and [7], the head movement mechanism (with some modification) given by [6], the two levels of syntactic structure (again with some modification) given by [5], and the mechanism for specifying meanings given in [2] and used throughout his later work. I have also used the modifications to standard theory given in [1], since they make the theory simpler and more suitable for processing and seem to be generally equivalent.

The module I built is somewhat incomplete, and does not handle phrasal movement, unknown open-class words (see below), or some types of null elements. These should all be possible to cover with an extension of my module, however, and are not included simply because there are a variety of examples that do not use them.

The module tries to keep four structures synchronized: a linear order of words, a linear order of clauses (complete noun phrases and complete verb phrases), a tree structure of phrases (clauses, prepositional phrases, and certain parts of clauses), and a conceptual structure for the meaning. As the process generally starts with only the first or the last of these, the module is also able to generate the other structures given one of the structures. In addition to being used to generate the other structures, each of these is used in determining other information.

This system does not use the traditional syntax derivation, which is to say that it does not realize the relations known as "movement" by starting with the "unmoved" positions and then changing the structures to produce "moved" structures. Rather, it starts with the positions it knows elements to be in (either as traces or overtly) and figures out the other positions the element must be present in. This method can handle all structures which involve the basic sort of movement (since it is mathematically equivalent to movement with traces), but not deletion or movement where order matters. As far as I can tell, all of the results explained by either of these[1] may also be explained in other ways. As the derivation model is currently more widely accepted, new analyses based on it are frequently produced, so the issue cannot yet be resolved.

The module works in several steps: first it splits the string into words, and then finds meanings for the words, including separating affixes. This process generates a sequence of morphemes. It could proceed in parallel, as the morphemes within a word are chained together, and the morpheme at the end of a word produces the morpheme after it by looking for the first morpheme in the next word; thus, the process of using the morphemes does not have to wait for all of the words to be separated into morphemes, or even seen.

Next the morphemes are built into cascades [5], which are linear orders of morphemes and other cascades; the embedded cascades are for noun phrases and embedded clauses. This seems to be the structure that binding domains are defined on, so

[1]in syntax; phonology has clear evidence of ordered transformations

it is needed for determining antecedents. However, these relationships are based on both the overt position of phrases and the trace positions, and phrasal movement is defined on a different structure.

From the cascade structures, layered structures are generated. These have prepositional phrases additionally made into data structures, instead of left in the cascade enclosing them. At this point, moved phrases could be identified, and their original positions filled in with traces. Adding the traces would, of course, alter the cascade structures as well, and thus change the possible binding relations, as linguists have found.

The back and forth interaction of the cascade and layered structures is significant, as it makes the fact that data propagates both ways explicit.

The layered structures are then used to generate conceptual structures. Each phrase gets a meaning, with phrases contributing to larger phrases in the positions chosen for their interpretation. At this point the result is evaluated for plausibility, and the structure may be rejected on purely semantic grounds, or because of yet more distant inferences.

Each of these transitions may run in either direction, so a conceptual structure can be made into the string. All of the same structural restrictions apply regardless of direction, so the strings it can generate are the same as the ones it can accept.

5.1 Morphology

There are two possible meanings for the word "morpheme". One refers to stems and affixes: the elements which are composed to form words. The other refers to the bits of information that syntax pushes around. In the common case, of course, these correspond, but there are exceptions. In the interests of keeping things straight, I will refer to the former simply as stems and affixes, and to the latter as *syntax items*.

For example, the word "foot" consists of just a stem, which corresponds to a single syntax item. The word "hands" consists of a stem and an affix, which correspond to another syntax item and the syntax item for plural nouns. The word "feet" consists, again, of a single stem, but it corresponds to the first syntax item and the syntax item for plurals.

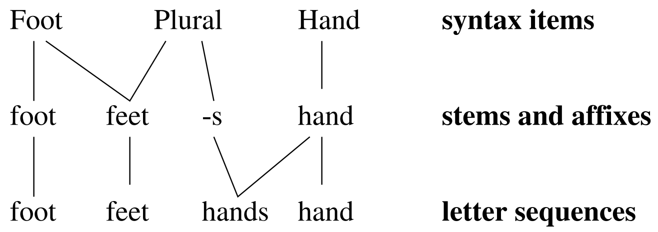

Figure 5-1: The composition of some words out of syntax items

It does not matter very much whether a group of syntax items is realized as a single stem or affix, as a single word composed of a stem and some affixes, or as separate words. Whether a syntax item is affixual or not do have syntactic effects, but these are present whether or not an actual affix is present in a particular use; the syntax item for the past tense has the same properties whether it is present as an affix on a regular verb, such as in “climbed” or corresponds only to part of a stem, as in “gave”.

A syntax item may not have any visible presence. For example, most of the present tense syntax items in English (except for the third person singular) do not change the verb form at all. In these cases, the syntax item may be detected in other ways, due to the structure of the sentence around it. Once again, there are syntactic effects which depend on whether the item is affixual or not, which can be used to discover the presence of the item and distinguish it from other items.

It is not usually particularly difficult to determine what overt affixes and stems make up a given word. The affix system is relatively powerful, in that it can have different effects depending on the form of the word affixed to, and can target not only the beginning or end of a word, but also such positions as before the stress in a multisyllable word (English slang) or the vowels in a word (Semitic). In addition to being able to add letters (or phonemes), it can replace letters or remove them.

The affix system is, in fact, exactly powerful enough that each possible affix can be inverted, such that it becomes instructions to the system for removing the affix.

This means that the system can take a word and remove affixes until it finds a known stem, and figure out the possible sets of stem and affixes for a word with mostly the same abilities that it takes to apply affixes to a stem to generate a word.

5.2 Cascades

Cascades are linear ordered sequences of heads (which are syntax items) and arguments (which are smaller cascades). For historical reasons, later elements in a cascade are referred to as "lower". Each clause and noun phrase[2] is represented at this level by a cascade, and other phrases may also be represented by cascades if they have been displaced by phrasal movement, but are ordinarily not represented separately.

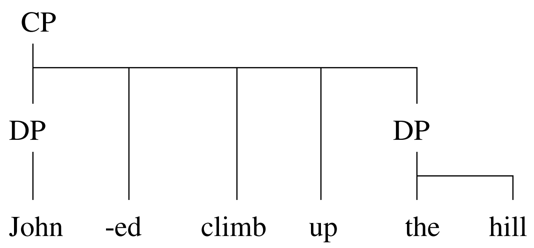

Figure 5-2: Cascade structure

Since language expresses many distinctions that we might like to treat as like words by variations in word order, it makes sense to use a concept of movement caused by special invisible items. This makes further processing much more uniform, as word order choices become identical to choices of actual words. As a simple example, the fact that questions begin with auxiliary verbs simple indicates that there is a "word" at that point which is "pronounced" by saying the auxiliary verb early. Additionally, some elements are pronounced as affixes on other words, and may have skipped over some other words to get to a suitable word. To the later stages of the module, it does not matter whether a given "word" is actually pronounced normally or not.

[2]In linguistics terms, it is not a noun phrase I mean here, but a DP. That is, a cascade could be "All the tea in China". "China" would be a cascade in the larger cascade as an argument.

The cascade representation keeps track of what elements are in what places and also keeps track of where the elements are pronounced. Having both versions is necessary for interfacing to representations on both sides, and is also necessary for determining whether the word order is unusual, and thus detecting invisible words.

The cascade representation also has the constraint that there are never two arguments without a head in between. This seems to be a constraint of natural language, keeping in mind that sometimes heads are invisible. According to some research the constraints on invisible heads and the requirement for heads between arguments explains a wide variety of effects. In my implementation, it makes the representation somewhat simpler to have arguments matched to heads.

Head movement processes seem to be based in a rather straightforward way on the cascade structure. Binding and coreference relations also seem to have their syntactic constraints at this level. This is, in general, a good point to pause in the process of interpreting sentences or producing them.

To produce cascade structures from a list of heads, my module moves from one end to the other, looking for heads which seem to go together. For example, in "Did John climb the tree?" it finds *John*, *Did ... climb*, and *the tree*. It then makes some decisions about what is included in what, and gets

[Did [John] climb [the tree]].

As it goes along, it notes a variety of invisible elements to get

[(Question) [(proper name, singular) John] (past) climb [the (singular) tree]].

To produce a list of words from cascade structure, the pronunciation information can simply be read out, applying affixes to stems as specified.

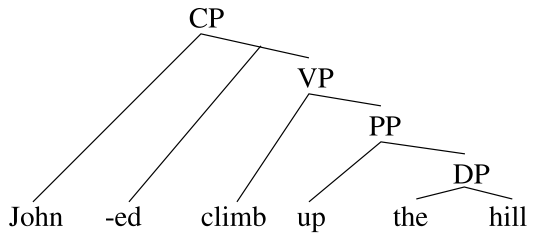

Figure 5-3: Phrase structure

5.3 Phrases

Cascade structures are not sufficient for determining some syntactic effects, and are not sufficient for computing meanings. Syntactic constituency (i.e., the units which may be repeated to ask for clarification and which can be moved about the sentence for focus and coordination) is not represented in cascade structures and has significant syntactic effects. Furthermore, prepositional phrases, which correspond to concepts in the meanings of sentences, are not cascades.

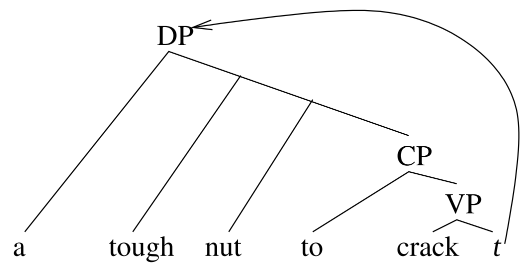

Figure 5-4: A phrase with "movement"

There is often a need to use a single argument in multiple ways. For example, in "A tough nut to crack", the nut is both the noun modified by the adjective *tough* and the object of *crack*. The phrase cannot be simply repeated, since that would refer to a different nut. Language could, in principle, use an overt marker to refer to the other use, but this situation is generally obvious from context. Natural languages, therefore, simply fail to pronounce the lower occurance of the element; I will mark this position for clarity with t. Pronouncing the higher occurance is better, because it can handle cases where there are two lower occurrences, as in "The book which I wrote t and she read t." This situation is therefore realized as movement.

The meaning carried by word order is relevant to this representation, as well. There are cases where an element is needed in a certain position for purely structural reasons. This is mediated by the properties of the heads, like the word order effects on the heads themselves. It has practical reasons, at least in some cases: word order variations could not be detected if the words did not move past other words. For English main clauses with auxiliary verbs, questions are distinguished by moving the auxiliary sooner; if a subject were not required, this could not be detected.

Converting between cascades and phrases is trivial. The main issue is that significant work gets done at each level determining additional information not present in the other form. In the case of making phrases, this involves determining the elements which are out of place or which belong in multiple places and determining the locations they should occupy. In cascade structure such elements have to fit in both places, and they participate in both places in effects based on cascade structure. Therefore, in the process of creating phrases to match a cascade, the cascade may be changed, as elements are found to be hidden in it.

Generating cascades from phrases involves figuring out the word order effects, and determining exactly what is pronounced where. It can also be crucial in word choice, since the use of pronouns depends in part on the structural relations revealed by cascade structure. This means that the process of generating a sentence from a meaning goes back and forth between representations significantly.

Phrase structure is the level at which word meanings work, in that words with arguments are with their arguments, and all of the portions of the sentence which have meanings are represented by individual structures.

Generating concepts from phrases involves, starting with the lowest head of the phrase, first applying to it all the higher heads, then taking the procedure thus generated and applying it to the required arguments, and finally factoring in the meanings of the other arguments, according to the rules for these.

Generating phrases from concepts requires, first, choosing those aspects of the conceptual structure which will be represented. Since much of the information in conceptual structure is represented by connections between concepts, and these connections, in general, span the entire space of this sort of knowledge the system has, and most of the connections may be presented in language, it is necessary to limit the sentence (for example, when referring to an object, you could, in principle, mention the entirely type hierarchy).

After that, the system must match the conceptual structures against word meanings to find the outermost phrase heads. Then the system works inward, mindful of the possibility of using pronouns or eliding phrases if permitted by the situation at various levels.

The product of the language system is a set of corresponding structures, from a sequence of letters or phonemes to a conceptual structure by way of several levels of syntax. It presents this information as another concept:

$$\begin{bmatrix} \text{SENTENCE} \\ \text{TEXT(“I shall remember...”)} \\ \text{MEANING} \left(\begin{bmatrix} \text{EVENT} \\ \text{...} \end{bmatrix} \right) \end{bmatrix}$$

The concept would carry additional information to express where it came from, such that the system could react to it appropriately in the context of the discourse.

5.4 The Lexicon

Language involves a substantial amount of arbitrary information. The connections between phonology and syntax and between syntax and semantics involve

The information of how to pronounce or spell a syntactic item, as well as irregular forms, must be listed. This forms a mapping between syntactic items and stems and affixes which is particular to a language.

Function words have properties which vary from language to language and are somewhat arbitrary. Whether an item is an affix or not and whether an affixes attracts a stem to it or moves to the stem's position are properties of the items. The system can be general across many languages with few global parameters, since the information may be localized in the functional syntactic items.

Each use of an item of certain types (nouns, verbs, prepositions, and modifiers) corresponds to a concept. The form of the concept is generally predictable from the item used, and normally involves use of the syntactic arguments in the phrase as arguments in conceptual functions of the concept. The mapping may introduce additional intermediate concepts, as well.

Some items involve a certain amount of variability in meaning. Some of this may come from invisible affixes, which may appear as a syntactic reflex of argument structure and be interpreted like other affixual heads in changing the meanings of items, but other cases seem to involve choices in how to connect items.

In order to represent the mapping between an item and the function of its arguments that it corresponds to, I use a structure which, following the usage of Jackendoff, I have called a Lexical Conceptual Structure. I have not used Jackendoff's linking theory [3], because I have found it easier to encode alternations by giving options for which concepts in the specification are replaced with arguments.

$$(a, a); (b, b)(c, \left[\ \textsc{Path}\ \right]);$$

$$\left[\begin{array}{l}\textsc{Event} \\ \textsc{Go}\left(\left[\ \textsc{Thing}\ \right]_a, \left[_{\textsc{Path}}\ \textsc{To}\left(\left[_{\textsc{Place}}\ \textsc{At-top-of}\left(\left[\ \textsc{Thing}\ \right]_b\right)\ \right]\right)\ \right]_c\right)\end{array}\right]$$

In using this item, the module attempts to unify each of the arguments which each possibility from the list at the appropriate position. Each record contains first the place in the conceptual structure the result would go and then the concept to unify with.

Chapter 6

Beyond

The system described in this thesis is only in the early stages, although hopefully the parts that have been done and parts that ought to be done first. I have tried to make sure the system would scale and that desirable new features would not force a rewrite of either the existing modules or especially the architecture.

I have thought of several directions that further research could go, and there are clearly other directions and other projects beyond each of the ones I have considered.

6.1 Processing Before the Language Module

One thing a flexible and error-correcting language module will do is allow modules that deal with the data beforehand to concentrate on their fundamental strengths. Since the language system is able to consider sentences and reject words that seem not to fit, it will be possible for a speech recognition module to generate a rough estimate of the speech, instead of a more careful analysis, and the morphology and syntax modules will reject words that are unlikely because they would not form a grammatical sentence.

Even if the module is working on typed text, it will probably have to handle typographical errors. People will generally understand text with a large number of shorthands, keyboard mistakes, and word mistakes, and will often not even notice that they read the text differently than it actually appeared. The ability to type in the way people type on IRC and have the computer expand or correct it to full correct sentences for applications which expect such would be very useful. As a simple case, it could act like a spellchecker, except that it would suggest words which let the sentence make sense to the systems before words that would not help it to understand the sentence.

Traditional speech recognition systems have tried to generate the correct words from the input using primarily statistical methods. This works based on a similarity metric on the phonological level, which has some relation to the higher-level information, in that the phoneme sequences in common morphemes in their appropriate contexts are more likely than other sequences, but it will not be able to recognize choices from widely separated parts of the input. In order to boost their effectiveness, they tend to prefer common real words, but this is not always helpful, because it is not generally effective unless the syntactic context, which enormously changes the likelihood of word appearances (consider, for example, articles, which appear very frequently, but do not ever appear in most positions in a sentence). So the system needs an estimation of the syntactic context, and so forth, as all of the information which applies to the situation is needed to distinguish reliably between similar sound patterns.

After all of this, the system produces a reasonably accurate sequence of phonemes. The human language faculty, after all, produces such a sequence accessible to consciousness, when listening to speech, and the perception of language is that it depends on this accurate sequence. However, what really happens is that the phoneme sequence is greatly affected by the further processing, which propagates its effects back to the phoneme level, changing the result that is available to consciousness; it is only the final revision that we experience, and thus have the illusion of magical accuracy that actually comes from trying extensively to use the possible outcomes.

Even if our hearing were perfect and there was no ambient noise, we still would not be able to necessarily distinguish differences in speech, because there are many possibilities which are entirely identical in actual production. For example, most word breaks are simply not present in the data. It is an effect of understand language that generates them in our experience, as can easily be seen by trying to listen for them in speech in a language one does not know; one begins to pick up the word breaks at the same time that one begins to understand the sentence structure, which makes sense because the problem is simply underdetermined beforehand.

Thus, it seems that human speech recognition proceeds by a large amount of guesswork and the later elimination of possibilities. The structure of the problem seems to require such an approach. Having a language module which performs an accurate syntactic analysis of the result of the phonology module with the ability to reject guesses at the point in the phoneme sequence where an error is likely, given the surrounding context is obviously required for this method.

6.2 Other modules

As was mentioned before, a vision system can generate the contact, support, and attachment relations that the language module uses. Adding such a system would be a relatively simple extension for the case where it is only producing new concepts bearing these relations.

A more ambitious addition would be a recognition system for tokens or instances of types based on [9]. Such a system would work to identify segments of the input with concepts in the conceptual module, so that other visual systems could produce new information for existing concepts. A simpler version of recognition, which might be additionally helpful, would simply associate positions in the image with concepts created by visual modules, such that multiple visual modules could contribute to the same concepts, mediated by a shared segmentation module.

A design which might be more helpful would involve the ability to generate a structure like a segmented image from purely conceptual data. This would allow the system to visualize what it is representing and to use its perceptual modules to make the sort of inferences which it would make from the image when looking at something even when an image is not available.

For all of these, the interface to language is through the conceptual system and the salience lists. Modules provide the information they judge significant and may try to answer queries from other modules at their own discretion.

It is likely that the system needs another central representation for representing the physical world. The representation is more of a model, is restricted to information about the physical world, and is not directly available to language. It provides the information relevant to pointing (especially pointing in the general direction of objects which are not visible). It would be shared between proprioception, vision, and sound-localization modules. It may be used for handling information from other internal modules (rather than in representing perception of the world), in order to imagine places one is not in. It maintains ties to the conceptual structure, containing, at least, PLACES and THINGS.

As mentioned in [2], the conceptual structure used for the physical world is reused for other fields, such as scheduling time, possession, and properties of objects. These, however, are grounded in their particular representations, and do not connect to the model particular to spatial representation. This is why it makes sense to talk about the stock market going up and down, but not about what the stock market would look like doing this. The conceptual module can treat the different fields similarly, as far as having a unified representation for things that move, and can therefore represent a variety of analogous relations on different fields with a single set of conceptual functions.

6.3 Learning

The collections of functional words and morphemes, including those which, in English are not distinguished on the surface but rather by their effects on word order, are large. However, even these are small compared to the number of open-class words and idiomatic phrases known to any English speaker. While the former set may be installed by a programmer considering each, there is no way a programmer could include all of the nouns and verbs, as new nouns and verbs are created constantly.

A certain quantity of words may be entered by hand. In fact, there is a set of words that are probably best handled that way. Function words tend to behave in odd ways which adults do not learn (for a new language) very easily; when they do learn such words, it is certainly not in the same way as new nouns and verbs. And even some nouns and verbs have properties that are hard to ascribe to novel words. It seems that such words are not open-class, despite being in categories to which new words may be added; these categories, therefore, contain limitations on the meanings of new words that are more strict than those on words of the categories in general. The words outside of the bounds of novel words in even open-class categories may be added in advance by the programmer.

Once a critical mass of words is achieved by manual insertion, however, the system ought to be able to learn new words from its interactions, if for no other reason than that one important sort of behavior is learning the meanings of technical terms.

The usual situation is that a piece of text will include a novel word, but will provide the same information using known words; the structure of the language allows the author to force both sets of information to apply to the same object. From this it can be inferred that the meaning of the novel word is at least compatible with the meaning of the known information, and thus the new word may be understood the first time it is seen.

Most likely the language module should generate open-class words of the appropriate sort when it encounters a word that it has no prior knowledge of. It should propose a minimal meaning for it (such as "a type of thing" or "affects"), and it should accumulate information that applies to the occurrences it has seen of the word. Based on these, it should be able to construct a meaning that it can keep. Some occurrences (like those which assume that the reader already knows the word) will contain little information (although often still a significant quantity), but others will be more helpful. In particular, the system could, if all else failed, try reading an online dictionary or asking the user for a definition.

More generally, it is very advantageous for the system to be able to deal with data that it understands very little and in only obscure ways. For instance, it is useful to be able to have a transitive verb with one of a few possible stems that involves starting uncontrolled movement, with no other information yet known. And it is useful to be able to deal with an object the system knows to have a major axis, but knows nothing else about, or an object that the system only knows was higher part way through moving across the field of view than the weighted average of its heights at the ends. Any of these pieces of information may be the limit of the systems knowledge about something, but any of them may be the only feature necessary to solve the problem at hand, and such information is useful for learning. [1]

[1] "John wugged Mary the ball" suggests to people that John did something causing the ball to move to Mary without his further intervention. "John pointed the spoo at Mary" tells people that the object involved has a forward end. Seeing an object move in the manner described above suggests that it was not supported.

Chapter 7

Summary

The Big Problems of AI: analogy, translation, ontology, creativity, and so forth, are all essentially equivalent, in that, if we had a solution to one that was really robust and complete, it would also be a good solution to the others. All of these require, as the main hard part, a model of general intelligence or common sense, and all of these, once they worked, could provide this facility to other systems.

At a lower level, however, there is another set of Big Problems, which do not get the attention that the former set do, since they do not involve results but, rather, methodologies. These include: module scopes, module interfaces, representation of cross-module information, and ambiguity. Any project which attempts to exhibit common sense will have to deal with these issues, but most projects do not consider their solutions as important results and do not present their architectural abilities and implementation details. Since the ability to extend a project depends on a good solution to these problems, and the ability to join multiple systems depends on them using the same solutions, these are some of the most important aspects of a system in determining its practicality.

In this project, I have focused on solving these problems and justifying my answers. I believe my system fulfills the first two requirements in a way suitable for general use. I have made suggestions for the third, but have no demonstration that they work. The fourth requirement is not, in my program, satisfied in a systematic way, and largely remains for future work. I believe that what I have done on an architectural level will not be invalidated by a solution to this omission.

My language system, while not complete, is sufficient to require the sort of support my architecture supplies and to show that it is possible to write a module using this architecture.

As it becomes desireable to have a more complete system than one wants to build at once and therefore to be able to use standard parts for those aspects that one is not directly interested in, having a common substructure and mechanism for connecting the pieces becomes vital. I have therefore examined what features such an architecture must have.

Bibliography

[1] Brody. *Lexico-Logical Form.* Number 27 in Linguistic Inquiry Monographs. MIT Press, 1995.

[2] Ray Jackendoff. *Semantics and Cognition.* Number 8 in Current Studies in Linguistics. MIT Press, 1983.

[3] Ray Jackendoff. *Semantic Structures.* Number 18 in Current Studies in Linguistics. MIT Press, 1990.

[4] Ray Jackendoff. *The Architecture of the Language Faculty.* Number 28 in Linguistic Inquiry Monographs. MIT Press, 1997.

[5] David Pesetsky. *Zero Syntax: Experiencers and Cascades.* Number 27 in Current Studies in Linguistics. MIT Press, 1995.

[6] David Pesetsky and Esther Torrego. T-to-c movement: Causes and consequences. In Michael Kenstowicz, editor, *Ken Hale: A Life in Language.* MIT Press, 2000.

[7] Steven Pinker. *Words and Rules.* Basic Books, 1999.

[8] Jeff Siskind. Visual event classification via force dynamics. In *Proceedings AAAI-2000*, pages 149–155, August 2000.

[9] Shimon Ullman. *High Level Vision.* MIT Press, 1996.