Anaphora and Ellipsis in Artificial Languages

Stephen G. Pulman
SRI International Cambridge Computer Science Research Centre[1],
and University of Cambridge Computer Laboratory.

Abstract

Artificial languages for person-machine communication seldom display the most characteristic properties of natural languages, such as the use of anaphoric or other referring expressions, or ellipsis. The paper argues that useful use could be made of such devices in artificial languages, and proposes a mechanism for the resolution of ellipsis and anaphora in them using finite state transduction techniques. This yields an interpretation system displaying many desirable properties: easily implementable, efficient, incremental, and reversible.

Linguists in general, and computational linguists in particular, do well to employ finite state devices wherever possible. They are theoretically appealing because they are computationally weak and best understood from a mathematical point of view. They are computationally appealing because they make for simple, elegant, and highly efficient implementations. In this paper, I hope I have shown how they can be applied to a problem ... which seems initially to require heavier machinery.
(Kay, [1987], p10)

1 Finite state languages

Since Chomsky [1957], most linguists have accepted that English, like other natural languages, is not a finite state language. Exactly where it falls on the hierarchy of languages defined by formal language theory is still a matter of debate, but there is virtual unanimity that natural languages are at least context-free, and probably mildly context-sensitive.

Although finite state languages and their associated processing algorithms are very important in many areas of computer science, especially in compiler technology (see, for example, Aho and Ullman [1977]) they have been largely neglected by linguists. There are some exceptions to this: recently, people have been interested in some finite state techniques for phonological and morphological description (Koskenniemi [1984]). Some computational linguists have also been interested in restrictions on context free processing that yield finite state systems (Church [1980], Pulman [1986], Pereira and Wright [1991]). But finite state languages in themselves are not usually regarded as very interesting linguistically.

It is nevertheless the case that many languages used by humans are finite state. For example, any panel by which some piece of consumer electronics is controlled

[1]Address for correspondence: SRI International, 23 Miller's Yard, Mill Lane, Cambridge CB3 1RQ. Email: **pulman@cam.sri.com**

implicitly defines a finite state language consisting of the set of valid inputs. (Of course, it would be possible for these control panels to define richer languages, but in practice this is unlikely to be necessary.) On my microwave cooker, for example, I can press 'high; 1; 30' to cook something on high for a minute and a half. Pressing these buttons in a different order will not achieve the same effect: perhaps no effect at all. This is the same as saying that there is a valid syntax with accompanying semantics which must be followed for communication to be successful. Video and cassette recorders, televisions, cookers and many other household and industrial devices define finite state languages of varying degrees of complexity in this way.

As cheap, small vocabulary speech recognition devices find their way into the marketplace, the range and flexibility of such artificial languages may well increase. It is not difficult to imagine many devices in the home or the workplace being controlled by spoken 'sentences' composed from a vocabulary of between 50 and 100 words. Under these circumstances an explicit grammar and semantics will be needed for successful use of this technology.

2 Reference, ellipsis and ambiguity

A characteristic of colloquial natural language is the frequent use of referring, anaphoric and elliptical expressions. In a phrase like:

Play it again, Sam.

the pronoun *'it'* refers to some contextually salient entity. In:

Sam played 'As Time Goes By' and then he played it again.

the pronouns *'he'* and *'it'* can be interpreted as anaphorically related to *'Sam'* and *'As Time Goes By'*. In a context in which Sam has just played *'As Time Goes By'*, an elliptical utterance like:

Again.

while not as quotable as the original, can be readily interpreted as meaning something like *'Play 'As Time Goes By' again, please'*.

Even from these few examples it will be obvious what function is served by these grammatical devices: the avoidance of repetition, and the ability to express a long message in a few words. There is a consequent burden on the listener, who must perform some operations to recover the contextually supplied part of the message, but in the majority of cases this process is carried out effortlessly and accurately. For example, it would, linguistically speaking, be perfectly possible to interpret 'it' as referring to the piano rather than to the song, but no-one does so.

A further phenomenon characteristic of natural language is lexical ambiguity (and other types of ambiguity, of course, although we shall ignore them here). A word like 'play', for example, has several different senses, even if we restrict ourselves to its occurrences as a verb: to play a game; play a musical instrument; play a joke or trick, etc. Again, with rare exceptions, this poses no problem for the listener as either the linguistic or the non-linguistic context will disambiguate.

Artificial languages generally do not use anaphora, ellipsis, or lexical ambiguity, and reference is usually achieved via the equivalent of proper names. This is because such languages are usually designed with the requirement that they be explicit and unambiguous, as well as easy to process. But there is no reason in principle not to design languages that do use such features, and it is in fact rather easy to imagine contexts in which they would lead to improved and more natural communication between man and machine. Consider for example, some kind of CD or cassette tape machine that is capable of selecting individual tracks, addressed by number. One might press a sequence of buttons 'play 2; play 4; play 9' in order to play those tracks. It would be much more economical to be able to say instead 'play 2; 4; 9', with the same meaning. Clearly, this is exactly analogous to the kind of ellipsis that is found in natural languages. As another example, we might have a sequence like: 'play 2; repeat'. 'Repeat' is a kind of anaphoric device here, a sentential proform meaning 'play it again', or, more generally, 'perform the last command again'. Furthermore, to cut down on the costs of producing the control panel, a manufacturer might well want to make certain buttons perform different functions in different contexts: thus a button marked $\gg$ might, in our example, sometimes mean 'forward', as in 'search $\gg$', and sometimes mean something like 'and all the numbers up to' as in '1 $\gg$ 5'. This behaviour is exactly analogous to the kind of lexical ambiguity found in natural language.

Other miniature languages that can easily be envisaged are illustrated in the following imaginary scenarios:

```
(operating system)
print 2 copies of foo.txt
mail it to jones
archive in txt.95

(computer game or robot control)
start engine1
run it for 10secs
stop
```

Of course, just as in natural languages, using these anaphoric and elliptical constructs might sometimes result in ambiguity. In the CD language, a sequence like 'play 2; 5; repeat' could be ambiguous between: 'play 2; play 5; play 2; play 5' and 'play 2; play 5; play 5', depending on whether 'repeat' is taken to refer to the shortest immediately preceding command, or the longest. I assume that the designer of a language like this will weigh carefully the balance between efficiency and ambiguity. Succinctness of expression is of no value if the result is a need for extra interactions to resolve ambiguity.

3 Processing of finite state languages

To be able to use a natural language with a computer we need some mechanism for parsing and interpreting the sentences of it. The process of parsing assigns syntactic structures to the input, which are then used as the basis first for compositional (i.e. context-neutral) semantics and then contextual interpretation. (See, for example, Alshawi [1992] for a description of a current state of the art system built along these lines.) Each of these stages is usually highly non-deterministic and difficult to carry out efficiently, since both the natural and formal languages involved are complex.

Finite state languages, however, can be processed very efficiently. Any finite state language can be described by a deterministic finite state automaton, or, often more succinctly, by a non-deterministic finite state automaton. In the case where the most succinct description is a non-deterministic one, algorithms exist for producing an equivalent (though possibly much larger) deterministic automaton. Algorithms also exist for transforming a deterministic automaton into the smallest possible equivalent automaton. (See Aho and Ullman [1977], Chapter 3).

Parsing and semantic interpretation of a finite state language may conveniently be thought of as a process of transduction between two finite state languages: the first being the original language, and the second being a language representing the context-neutral semantic interpretation of the input sentence. (In the case of the approach to NLP described in Alshawi [1992], this is the equivalent of the 'quasi-logical form' level, in which the basic predicate argument structure of a sentence is represented, but contextually dependent aspects of interpretation (pronouns, definites, ellipsis, tense etc. are left unresolved). Since the syntactic structure of finite state languages is essentially trivial, once we know the sequence of states that is involved in recognition we can proceed directly to the meaning of the input sentence.

Finite state transduction is again a well-understood technology (Aho and Ullman [1972]: p223; Kay [1984]; Ritchie et al. [1992]; Kaplan and Kay [1994]). A transducer can be thought of either as a machine which reads two input tapes and then makes state transitions depending on what is written there, or alternatively as a machine which reads an input tape and writes an output tape depending on the state transitions that are possible. However, for the designer of a language like those we are talking about it is tedious in the extreme to have to write such low level transducers directly. It is much better to be able to have a relatively high level notation in which the properties of the language can be expressed at an appropriate level of abstraction, and then produce the corresponding transducers by a process of compilation.

There are various ways of achieving this. One way, developed originally for a phonological notation, is to use an automatic compiler for a high-level rule formalism. (See Ritchie et al. [1992], and Kaplan and Kay [1994]. Kaplan and Kay describe a 'regular relations calculus' which provides a set of tools for the rapid construction of compilers for a range of such notations.). Another way, which we shall adopt here, is to write the grammar for the language as a restricted form of push-down transducer (the transducer equivalent of a context-free system, or recursive transition network: Aho and Ullman [1972], p.227). This allows us to write succinct and natural grammars, while the restrictions we shall impose will still ensure that the device remains finite state in capacity. The simplest restriction that will ensure this is to require that nesting of subnetworks (and hence recursion) is limited to some small depth. For our example language this depth can be 2. Another notational convenience is to allow an extra limited 'memory' for the output tape. The primary purpose of this in our example language is to allow for alternative syntactic permutations to be transduced into the same output. Provided this memory is limited in capacity the device overall will remain a finite state one overall, and can be compiled out automatically into something that is directly finite state (though much bigger).

4 An example language

To illustrate, assume that the language we are dealing with is that of a controller for a music cassette or CD playing and copying system of some kind. The player can locate tracks identified by number, play them, erase them, or copy them to an output recording device. (Ignore the question of whether the example is commercially realistic: it is probably not. However, it has been fully implemented, along with a graphical 'control panel' interface, and a simulation of the CD-like back end device.)

The system can be instructed by commands such as:

```
erase 1 >> 4                ; erase tracks 1 to 4
play 4 << 2                 ; play tracks 4 to 2 in reverse order
search >>                   ; search forward
play 1 play 3               ; play track 1 then play track 3.
1 2 3 play                  ; play track 1, play track 2, play track 3.
play 1 3 repeat             ; play track 1, play track 3, then repeat (both).
```

We shall assume that the syntax and compositional semantic interpretation of this language can be given by the following grammar, which is written in a context-free like notation but defines a finite-state language. In the following example, non-terminal symbols are given in upper case, and terminals are lower case symbols, numbers, and ';', used as an end of sentence marker.

Alternative right hand side expansions of the non-terminal are indicated by the |. The portion of the rule after ':' indicates the semantics assigned to the structure by the rule. In this part of the rule a number refers to the interpretation of a daughter consituent, referred to in left to right order. The interpretation of terminal symbols is assumed to be the symbol itself. Thus for example rule 8 says that the nonterminal 'DIRECTION' can be expanded as either $\ll$ or $\gg$ and that the semantics assigned by the rule is the interpretation of the first (only) daughter, whatever that is.

1	COMMAND	$\rightarrow$	ACTION	:	1
2	COMMAND	$\rightarrow$	ACTION COMMAND	:	1 ; 2
3	ACTION	$\rightarrow$	repeat	:	1
4	ACTION	$\rightarrow$	ACT TRACKS	:	1 2
5	ACTION	$\rightarrow$	TRACKS ACT	:	2 1
6	ACTION	$\rightarrow$	search DIRECTION	:	1 2
7	ACT	$\rightarrow$	play| copy| erase	:	1
8	DIRECTION	$\rightarrow$	$\ll \mid \gg$	:	1
9	TRACKS	$\rightarrow$	TRACK-SPEC	:	1
10	TRACKS	$\rightarrow$	TRACK-SPEC TRACKS	:	1 ; action 2
11	TRACK-SPEC	$\rightarrow$	NUM	:	1
12	TRACK-SPEC	$\rightarrow$	NUM DIRECTION NUM	:	1 2 3
13	NUM	$\rightarrow$	1| 2| 3| 4| 5	:	1

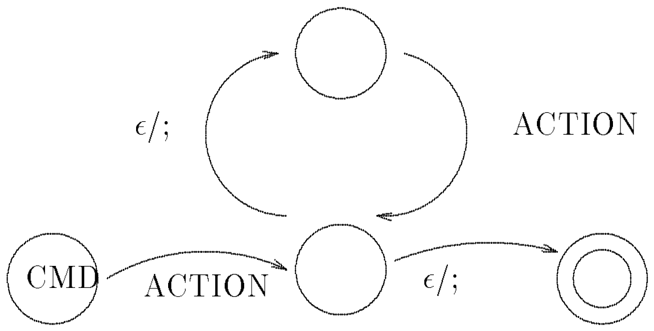
Some of these rules may need further explanation. Notice that rules 4 and 5 allow for the same components to be expressed in different orders, but leading to the same interpretation. It is for this kind of rule that the limited memory for the transducer is used: although it would be possible to write a more complex transducer with no memory to achieve this effect, a large number of extra states and transitions would be required. It is simpler to just 'store' the interpretation of some constituents in memory and recall them (in the same order) at the appropriate position.

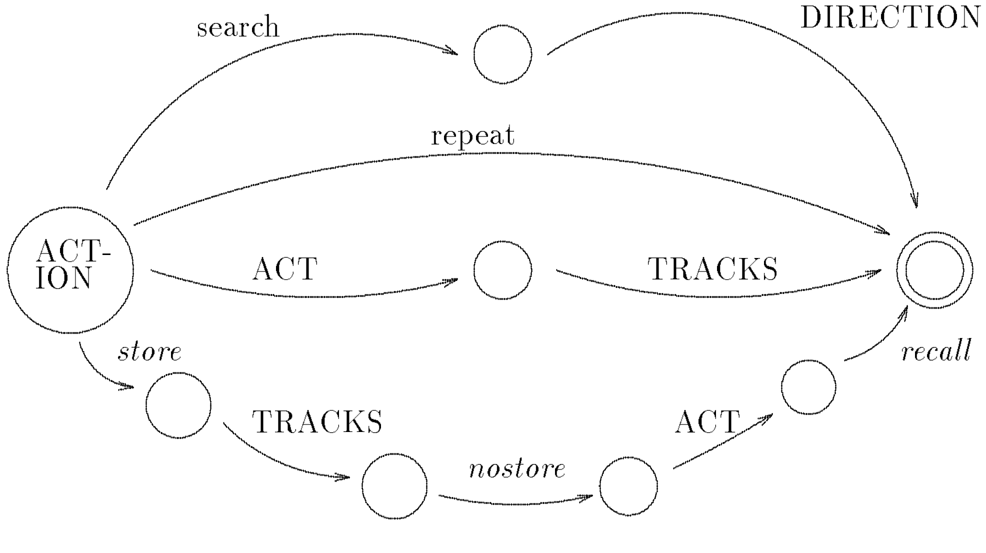
Rule 10 inserts in the interpretation a marker 'action' which can be thought of either as a 'proform' (like 'repeat') or as a marker telling the contextual resolution component that there is an ellipsis to be filled in here. (Many current linguistic theories postulate entities like 'empty pronouns' for phenomena that are like ellipsis). The DIRECTION terminal symbols are introduced into two different contexts, and their interpretation will vary according to the context they are found in. This simple example language then contains an instance of each of the devices characterised above as typical of natural languages: anaphora, ellipsis, and lexical ambiguity.

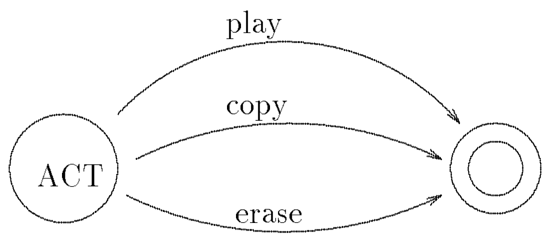
The above grammar is given for illustration. The information it represents can be equivalently, though less perspicuously, encoded in a finite state transducer, augmented as described earlier. (Although this was not done in the implementation, it would be possible to compile a grammar into such a transducer automatically, and this is the envisaged mode of construction for such systems). An ordinary finite state transducer is just like a finite state network except that transitions are labelled with pairs of symbols, α/β, representing the input and output tapes. We allow the empty symbol, ϵ, meaning that the input or output tape is not advanced at that point. If a single terminal symbol labels a transition the interpretation is that the output is the same as the input, i.e. a single terminal symbol α is interpreted as α/α.

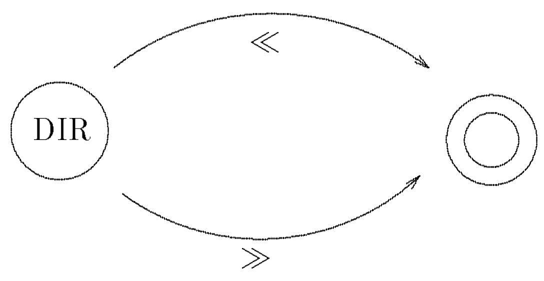
Our simple augmentations can be achieved by adding several new kinds of transition. One kind of transition from a state S_i to S_{i+1}, labelled with the name of a subnetwork, will have the obvious interpretation that things proceed as if S_i was the initial state of the subnetwork and the final states of that subnetwork lead back to S_{i+1}. A transition between states labelled 'store' will have the effect of starting to write the output tape to a 'memory' and a transition labelled 'nostore' will stop this happening. A transition labelled ‘recall’ will write the contents of the memory onto the output tape, and clear the memory. It should be clear that provided calling of subnetworks is restricted, such a transducer is just a more convenient notational equivalent of a simple finite state transducer.

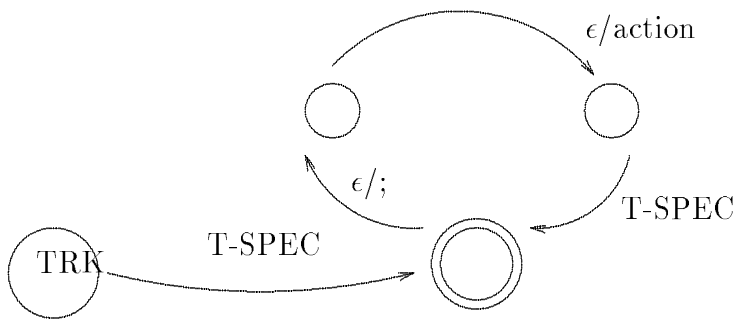
Given these notational conventions the language defined by the grammar above can be equivalently represented by the following transducer. The CMD network corresponds to rules 1 and 2; ACTION corresponds to rules 3-6; ACT and DIR to rules 7 and 8 respectively; TRK to rules 9 and 10; T-SP to rules 11 and 12 and NUM to rule 13. In each network the initial state is the leftmost.

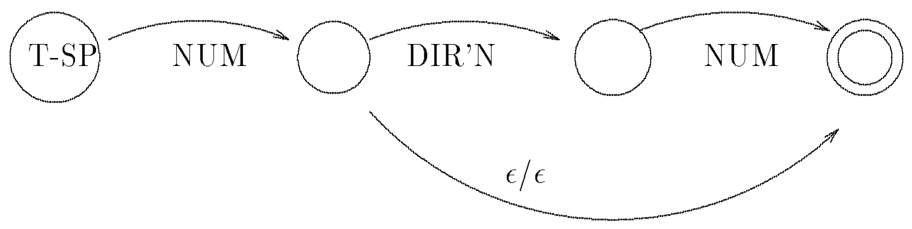


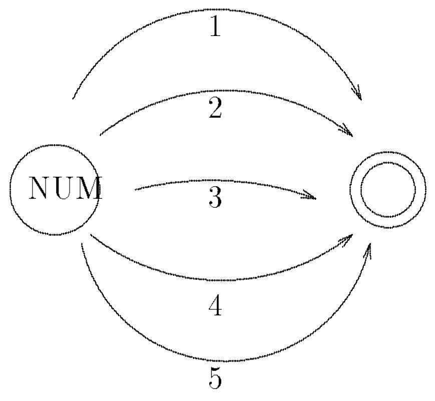



play
copy
ACT
erase


≪
DIR
≫







Some representative inputs and outputs will be:

play 1 repeat	⇒ play 1 ; repeat ;
play 1 2	⇒ play 1 ; action 2 ;
1 2 3 play	⇒ play 1 ; action 2 ; action 3 ;
search ≫	⇒ search ≫ ;
erase 2 ≫ 4	⇒ erase 2 ≫ 4 ;

5 Contextual resolution

Now we turn to the process of contextual resolution of the sentences of our artificial language. The transducer just given carries out the equivalent for our language of syntactic and compositional semantic analysis, giving us a meaning representation that is independent of the context of the sentence. That representation contains various constructs representing the meaning of contextually dependent items: in particular, 'repeat' and 'action', and some whose interpretation depends on the linguistic context, such as '$\ll$' and '$\gg$'.

We shall assume that the process of contextual resolution of a sentence containing one or more of these constructs consists of transducing it to another sentence in which no such constructs appear and which can thus be interpreted directly. In this respect our language probably differs quite significantly from natural language: it is an open question whether every sentence of a language like English can be easily paraphrased by others containing no referential or elliptical devices (except perhaps for names), but there is some evidence that the answer is in the negative. For example, elliptical comparative constructions like 'Paris is nearer to London than Rome' can be paraphrased into a pair of non-elliptical sentences conjoined by some comparative marker: 'Paris is near to London *to a greater extent than* Paris is near to Rome', and 'Paris is near to London *to a greater extent than* Rome is near to London', but some apparently similar sentences like: 'Show me a company with higher profits than IBM' do not seem able to be paraphrased into smaller non-elliptical complete sentences in this way: '???Show me a company with profits *to a greater extent than* IBM is with profits'. Analogous examples with pronouns are well known: 'the boy who deserves it will get the prize he wants' cannot be paraphrased so as to eliminate all pronouns or other anaphoric devices. It is thus reasonably clear that the techniques described here would not extend to other than simplified fragments of a natural language, and they are not offered as an account of 'real' anaphora and ellipsis processing.

The fact that we can characterise our resolution task as one which takes an input sentence (containing certain constructs) and gives another output sentence (with various kinds of replacements for these constructs) means that we can regard it as another type of finite state transduction operation. However, things are a little more complex than they have been so far, for we need to take account not just of an input sentence, but also of a context. To keep things simple, we shall assume that our notion of context can be adequately modelled by a sentence of our language, one containing no unresolved items and which, we shall assume, represents the contextually resolved interpretation of the last command processed. We use the sentence boundary marker ';' (which is introduced by the analysis transducer) to reset the context to the last contextually resolved atomic command, where an atomic command is one delimited by this marker. Other choices for the scope of the context are possible and richer notions of context can be modelled by sets of such sentences, with an extension to the basic mechanism, but we shall keep things simple for the purposes of illustration.

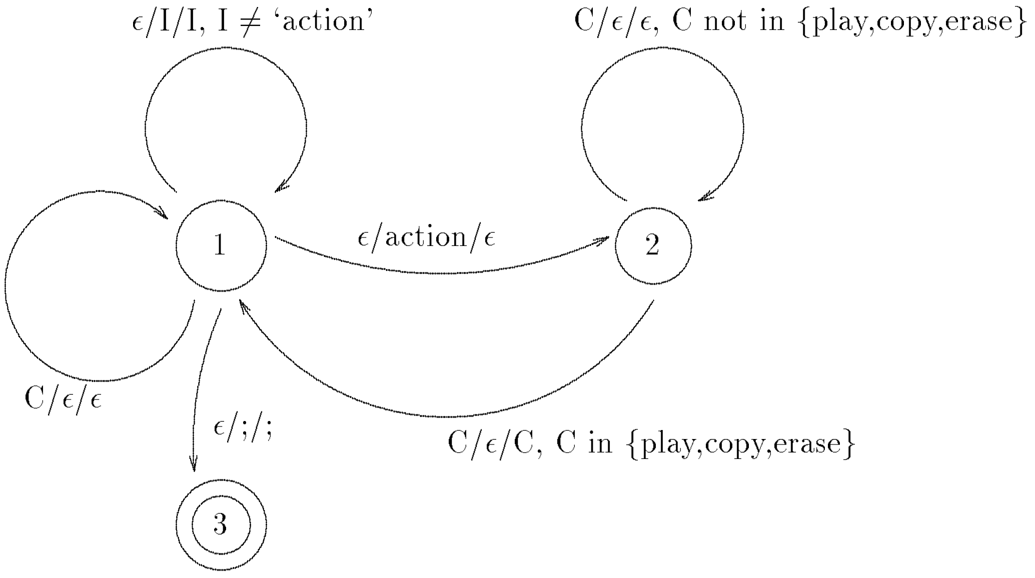
We shall need to model the contribution of each particular context-dependent element. We shall assume that the rules governing the interpretation of such elements can be expressed as instructions to carry out a certain type of transduction. These transducers will be slightly more complex than the previous type because of the need to take into account two inputs in order to produce an output: the unresolved sentence and the context sentence. We need a multi-tape transducer, therefore.

Multi-tape transducers were introduced into computational linguistics by Kay

[1987]. They are the obvious extension to simple transducers, allowing multiple input tapes and one output tape. The acceptance conditions for multitape transducers are analogous to those for ordinary transducers (see Kaplan and Kay [1994]). For a transduction to be successful we must have exhausted all input tapes and be in a final state. We can describe them informally in the usual diagrammatic way, except that transitions will now be labelled with n-tuples (here, triples) of symbols instead of pairs.

The strategy we shall adopt is to model contextual resolution as a transduction from the unresolved sentence and the context to a resolved sentence. We will write specialised multitape transducers for each different type of item requiring resolution. (We assume that some higher level notation undergoing compilation would actually be used.) These transducers will then be combined by disjoining them into one large transducer.

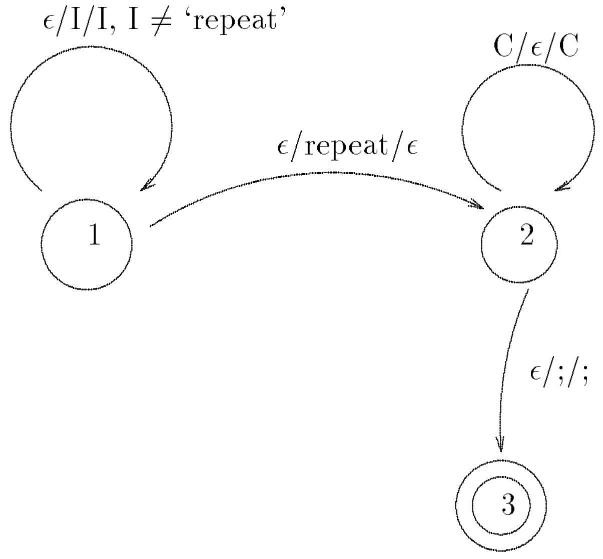
To illustrate, we give a transducer which, given an input sentence like 'action 1' and a context like 'play 3' will resolve the 'action' construct to 'play', producing as output 'play 1'.



ε/I/I, I ≠ 'action'
C/ε/ε, C not in {play,copy,erase}
1
ε/action/ε
2
C/ε/ε
ε/;/;
C/ε/C, C in {play,copy,erase}
3

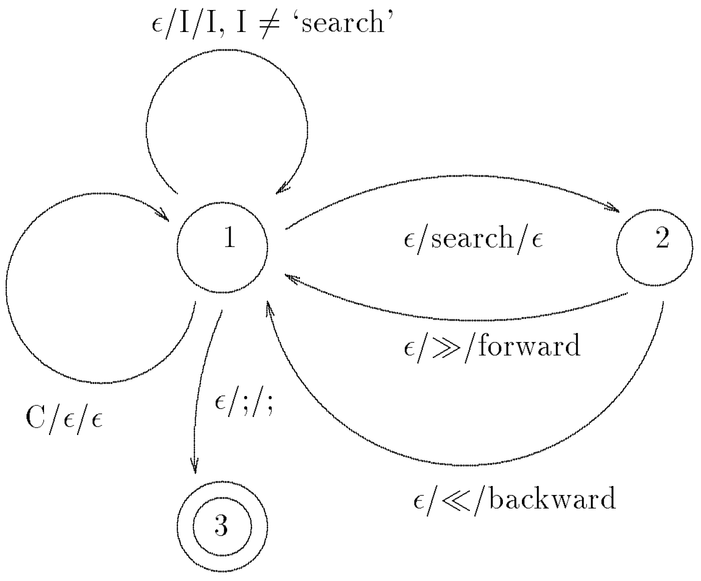
There are just three states to this transducer. Transitions are labelled *Context/Input/Output*, with variables over symbols and associated conditions, for readability. (These can all be expanded out to atomic symbols in a real implementation.)

In state 1, the initial state, the looping transition labelled $\epsilon/I/I$ simply echoes the input tape to the output tape, while not advancing the context tape. This allows for those symbols that do not need resolving (by this transducer) to be simply echoed to the output tape, and analogous transitions will be needed on all our resolution trandsucers. When the symbol 'action' is encountered on the input tape, neither the context nor the output tapes are advanced, but we move to the second state of the transducer via the transition labelled $\epsilon/action/\epsilon$. In this state, the looping transition labelled $C/\epsilon/\epsilon$ moves through the context tape without advancing either of the other two tapes, until a candidate for 'action' is encountered. This allows the transition labelled $C/\epsilon/C$ to be taken, which writes the referent of 'action' on the output tape, and returns us to state 1. Now the earlier looping transition can be taken to copy the remainder of the input tape to the output. Any remaining context can be consumed by the other looping transition labelled $C/\epsilon/\epsilon$. The remaining transition is triggered by the end marker, taking us to the final state.

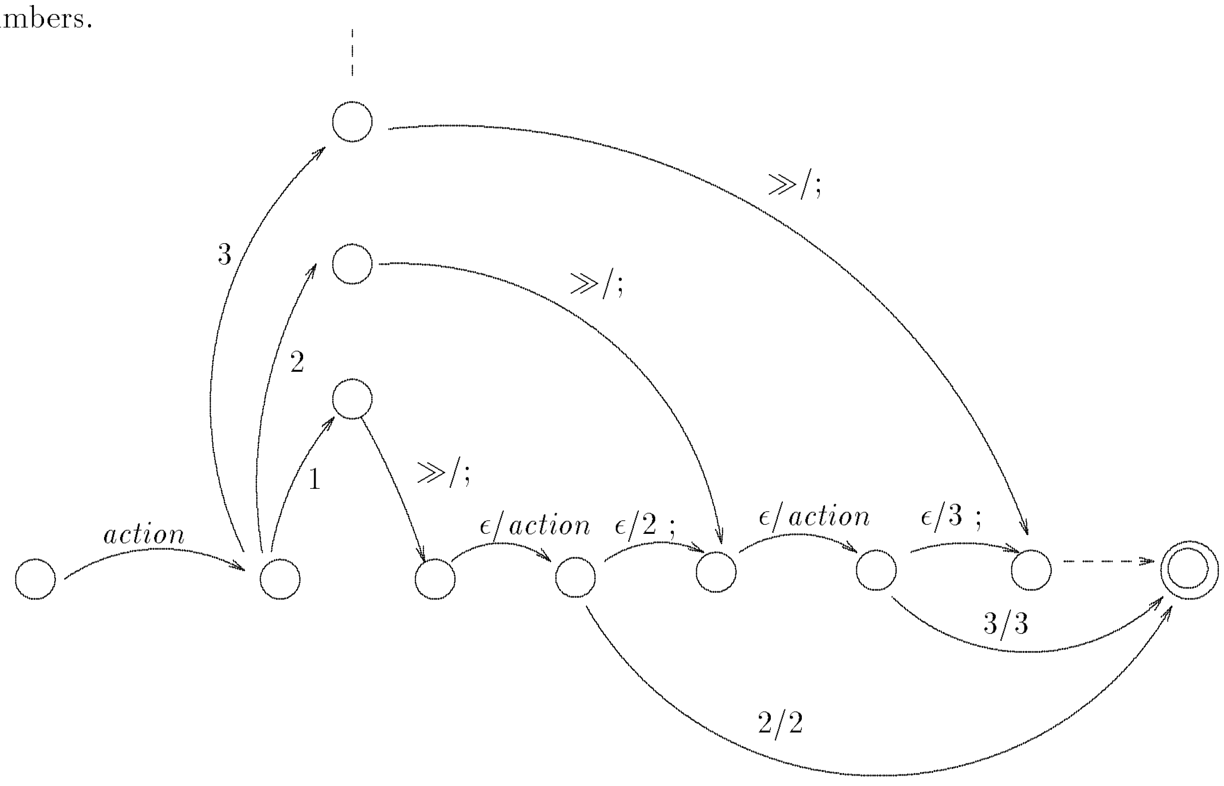


This second transducer illustrates the interpretation of a proform. It replaces a command consisting of 'repeat' with the contents of the context tape. The looping transition in the initial state 1 ignores any other input. If 'repeat' is encountered we move to state 2, and the looping transition there copies the context tape to the output. The remaining transition picks up the end marker and goes to the final state.

The next transducer is rather similar to the first. It resolves the lexically ambiguous words $\ll$ and $\gg$ to the symbols 'backward' and 'forward' respectively when they occur in the context of 'search'. However, no important reference is made to the context tape, since this resolution depends entirely on the input sentence itself. In this respect, this example is perhaps not well chosen, as the same effect could be achieved by writing the initial parsing transducer in the appropriate way. However, some simple extensions of the language would require real ambiguity resolution: consider if we allowed a symbol like $\gg$ to appear alone. Then it might be interpreted as something like '(search) forward' in a context preceded by 'search backward', or as '(play) 3' when following 'play 2'. Admittedly, this is perhaps beginning to cross the boundary between ambiguity resolution and reference resolution: in 'real' English the boundary is similarly often rather unclear. For example, the process of interpreting 'have' (as in 'have no money, or 'have measles') is more akin to pronoun resolution than to sense disambiguation.



The final transducer that we shall consider interprets ≫ when it occurs between numbers.



State 1 is initial. Again the context tape is irrelevant and in the interests of readability it, and one or two extra transitions it requires, are omitted.

Here the 'action' label stands for any of the actions: separate versions of this transducer will be needed for each of them. The operation of the transducer is as follows: when a sequence like 'play 1 ≫ 4' is encountered, the obvious sequence of transitions for 'play' and '1' are taken. Then the transition for ≫ emits a boundary symbol and takes us into a state where an empty symbol for the input tape is paired with another 'play' on the output tape. Then the next number in the sequence is produced on the output tape. This can happen in either of two ways. If the next symbol on the input tape is that number, the transition labelled 2/2 can be taken, leading directly to the final state of the transducer. Otherwise, an empty symbol on the input tape is paired with a 2 on the output tape, immediately followed by another empty symbol on the input tape paired with a boundary symbol. (In the diagram this sequence has been compressed into one transition labelled '$\epsilon/2$;' for reasons of space.) We then go into a state where this sequence of events is repeated for the next number in the sequence. The dotted lines indicate that other states and transitions in the sequence should be assumed.

With this transducer, our input 'play 1 $\gg$ 4' will be resolved as 'play 1; play 2; play 3; play 4'. Transducers for the interpretation of $\ll$ will be analogous except that the numbers will be in the reverse order.

6 Composition of transducers

The transducers we have just sketched collectively cover all the cases of contextual resolution that occur in our artificial language. We can use them to effect this resolution by combining them into one large transducer, which will take the output of the parser and produce the contextually resolved forms. The parallel (disjunctive) composition operation is simple, and is essentially the same as that given in Aho and Ullman [1977], p.96 for union of finite automata. First we identify all the start states of the original transducers. Then, from all the final states of the original transducers we create a transition labelled ϵ/ϵ to a single new final state. Although transducers containing ϵ are not closed under complementation and intersection, finite stateness is preserved under this operation.

It is not decidable whether two non-deterministic finite state transducers are equivalent (Aho and Ullman [1972], p.237) so in the general case it is not possible to adapt the familiar algorithm for minimisation of automata (Aho and Ullman [1977], p.101) to shrink the resulting machine. Nor is it always possible to determinise a non-deterministic transducer. Nevertheless, some obvious simple techniques like eliminating duplicated transitions, or identifying adjacent states connected only by a transition labelled ϵ/ϵ will serve to simplify the resulting automaton to a significant extent.

A final step is to compose the transducer we use for parsing with that we have now obtained for resolution. In order to do this we first need to transform the convenient notation we used earlier, which involved subnetworks and a memory, into the lower level direct transducer notation we have been using for the resolution component. This transformation is simply achieved by the following (ordered) rules:

1. Starting with the basic network, recursively replace each transition labelled with the name of a subnetwork by a copy of that subnetwork. In most cases this can be simply done: given a transition from A to C labelled B, we identify the start state of subnetwork B with state A and its final states with state C. But if there are transitions leaving a final state

of subnetwork B, or other transitions entering C, it is necessary instead to add transitions labelled ϵ/ϵ from those states to state C, otherwise the language accepted will be changed.

2. For each sequence of states begun by 'store' and ended by 'nostore', replace transitions labelled α/β by transitions labelled α/ϵ. Remove the 'store' and 'nostore' transitions, and rename states appropriately.

3. Replace each corresponding 'recall' transition by a sequence of transitions and states labelled ϵ/β, where the values of β are as in rule 2.

Rules 2 and 3 can be illustrated by considering a set of states and transitions as follows:

From	Label	To
1	store	2
2	a/a	3
3	b/b	4
4	nostore	5
5	c/c	6
6	recall	7

This transducer maps 'abc' to 'cab'. By applying rules 2 and 3 the resulting compiled transducer will be:

From	Label	To
2	a/ϵ	3
3	b/ϵ	4
4	c/c	6
6	ϵ/a	6.1
6.1	ϵ/b	7

This accepts exactly the same input and output language as the original.

Now we can compose the parsing transducer with the resolution transducer. First we will define serial composition of ordinary transducers: the operation we need is a simple variant of this. The serial composition operation proceeds as follows (Kay [1984], p.112). It assumes that the output language of the first transducer is identical to the input language of the second transducer. The states of the new transducer will be a subset of the product of the states of the two transducers, labelled accordingly.

Beginning with the initial states of the two transducers, A_1 and B_1, recursively construct new states and transitions according to the following rules:

if there is a transition from A_i to A_j labelled α/β, and one from B_i to B_j labelled β/γ, then make a new transition labelled α/γ from $\langle A_i,B_i\rangle$ to $\langle A_j,B_j\rangle$.

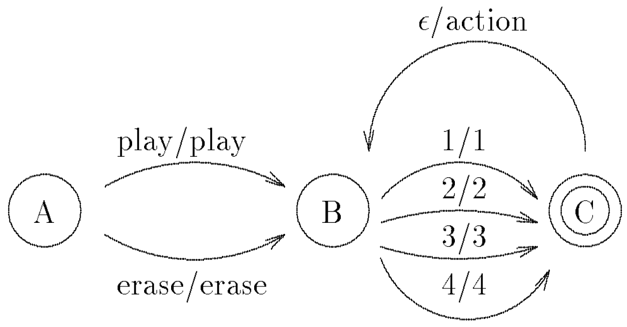
if there is a transition from A_i to A_j labelled α/ϵ, and one from B_i to B_j labelled β/γ, then make a new transition labelled α/ϵ from $\langle A_i,B_i\rangle$ to $\langle A_j,B_i\rangle$.

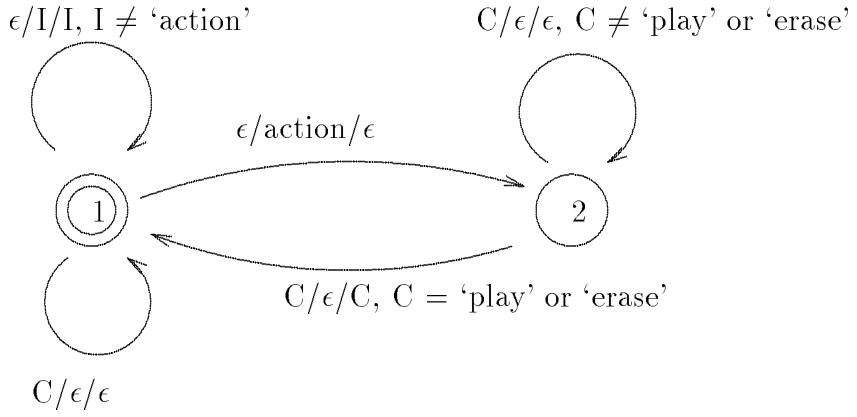
if there is a transition from A_i to A_j labelled α/β, and one from B_i to B_j labelled ϵ/γ, then make a new transition labelled ϵ/γ from $\langle A_i,B_i\rangle$ to $\langle A_i,B_j\rangle$.

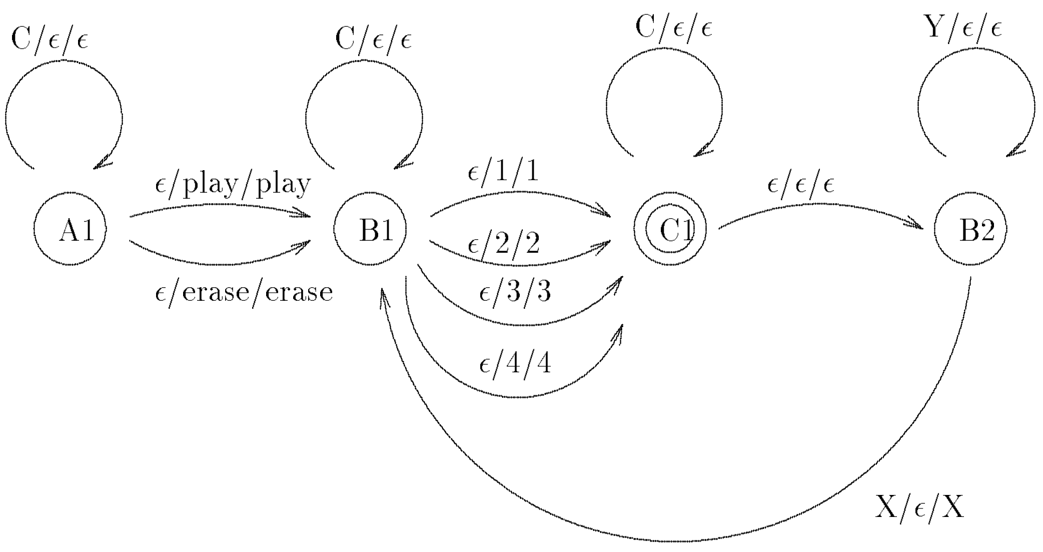
Transitions from constructed states $\langle A_n, B_m \rangle$ are derived from the transitions leaving the component states according the above rules. New states $\langle A_n, B_m \rangle$ are final if and only A_n and B_m are final.

In our case we are composing an ordinary transducer with one input tape of a multitape transducer. Other than something to keep track of the symbols on the context input tape, the same technique can be used.

To illustrate this process we give first of all two input transducers. The first is a simplified version of part of the transducer given earlier and the second is a simplified version of the multi-tape resolution transducer for the 'action' construct. The third transducer is the result of the serial composition of the first and second. The output tape of the first transducer is composed with the sentence input tape of the second transducer.







In this transducer, X must not be 'play' or 'erase', and Y must be one of them. Notice that the 'action' construct has disappeared completely in the operation which led to the transition between C1 and B2. (This transition could in fact be eliminated, merging C1 and B2. Also, the looping transitions in A1 and B1 will never be used and could be eliminated. In general the results of composition may well be capable of being simplified considerably by the various methods described earlier.)

7 Conclusions

We have now outlined the steps by which a system for the compositional and contextual interpretation of certain types of artificial language can be constructed. The resulting system consists of a multitape transducer with two input tapes and one output tape. One input tape represents the current context, and the other the current input sentence. The output tape represents the interpretation of the sentence in that particular context.

There are several practical advantages to this approach to contextual interpretation, apart from its conceptual clarity.

First, composing into one large transducer automatically gives us a large measure of what is often called 'incremental interpretation'. That is to say, semantic and contextual effects are computed as the input is being processed, rather than waiting for the sentence to be completed and submitted to subsequent stages, as is usually the case in full-blown NLP systems. This means that it is possible to take actions or give informative feedback to the user before the input is complete. Some qualification to this has to be made for local non-determinism which may be present, some of which can arise from the compiling out of the limited 'memory' used in the high level notation.

Second, composition into one transducer also means that it is trivial to compute possible continuations of incomplete messages. This also can give useful feedback to the user, or in cases where speed of response is of the essence, useful forewarnings to back end systems.

Third, and for the same reason, it is simple to suggest corrections or repairs for messages that are not within the finite state language defined by the system.

Fourth, it is easy to associate probabilities with transitions. In the case where a sentence is ambiguous, either inherently or because multiple resolutions are possible, the most likely reading can be signalled as such. It is simple to assign probabilities automatically given a corpus of examples, or allow them to be adjusted dynamically for a particular user over time.

Fifth, transducers are reversible, allowing for context-dependent generation of messages to be achieved with no further effort. (However, it should be remembered that non-determinism can be an issue here. A transducer may be deterministic in one direction, but not in another. A transducer that is highly non-deterministic may not be very efficient.)

Lastly, but by no means least, finite state transducers can be implemented very simply and efficiently. In the case of consumer devices, where simplicity and robustness are paramount, this is a great advantage.

Although the language used here for illustration is extremely simple, leading to transducers with only a few dozen states, the technique would scale up to somewhat more complex examples. Provided that a suitable high level notation is used, the size of the transducers themselves is unlikely to be a limiting factor, within reason: experience with transducers in computational morphology has shown that 'natural' systems do not approach the worst case of time and space behaviour that complexity results would suggest (Kartunen, Kaplan and Zaenen [1992]; Koskenniemi and Church [1988]).

There remain a few further things to explore. The example system described above (a version of which has been fully implemented) can only deal with one contextually sensitive item per atomic sentence, because the resolution transducers are composed in parallel with each other rather than in series. The grammar is defined so as to ensure that this limitation is always observed. But it is easy to imagine circumstances in which this would become tedious. If the limitation is to be relaxed there are two options which could be taken. Firstly, one could simply keep parsing and resolution separate and reapply the resolution transducers until all contextually sensitive elements in the input had been resolved. However, this would lose some of the benefits of serial composition of parsing and resolution, namely reversibility and incrementality.

The second alternative would be to produce a serial composition for each ordering of each subset of the resolution transducers, putting these in parallel composition and then serially composing the parser with the resulting (presumably rather large) resolution transducer. It remains to be seen whether in realistic applications this procedure would result in final transducers of manageable size. But when considering what is a 'manageable size' we should perhaps bear in mind that there is no practical reason for transducers with many millions of states not to be part of your toaster or coffee machine, even at the current state of the art.

8 Acknowledgements

The work reported here was supported at SRI by a contract from Philips Research and the IPO, Eindhoven. I am grateful to Don Bouwhuis, David Carter, Kees van Deemter and Reinder Haakma for their comments on an earlier draft, and to my referees for suggesting improvements in the paper.

References

[1992] Alshawi, H. (ed.) 1992. *The Core Language Engine*, MIT Press: Boston, Mass.

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

[1977] Aho, A. V. and Ullman, J. D. 1977. *Principles of Compiler Design*, Addison Wesley: Reading, Mass.

[1957] Chomsky, N. 1957. *Syntactic Structures*, Mouton: The Hague.

[1980] Church, K. 1980. On Memory Limitations in Natural Language Processing, Technical Report, MIT: LCS/Tr-245.

[1994] Kaplan, R. and Kay, M. 1994. Regular Models of Phonological Rule Systems, *Computational Linguistics* 20:3, pp. 331–378.

[1992] Karttunen, L., Kaplan, R., and Zaenen, A. 1992. Two-Level Morphology with Composition, in *COLING-92, Proceedings of the 14th International Conference on Computational Linguistics*, pp. 141–148.

[1984] Kay, M. 1984. When meta-rules are not meta-rules, in K. Sparck Jones and Y. Wilks (eds.)*Automatic Natural Language Parsing*, Ellis Horwood: Chichester, pp. 94–116.

[1987] Kay, M. 1987. Nonconcatenative Finite State Morphology, *Proceeedings of the Third European Conference of the Association for Computational Linguistics*, ACL: Copenhagen, pp. 2–10.

[1984] Koskenniemi, K. 1984. A general computational model for word-form recognition and production, in *COLING-84, Proceedings of the 10th International Conference on Computational Linguistics*, pp. 178–181.

[1988] Koskenniemi, K. and Church, K. 1988. Complexity, Two-Level Morphology, and Finnish, in *COLING-88, Proceedings of the 12th International Conference on Computational Linguistics*, pp. 335–340.

[1991] Pereira, F. C. N. and Wright, R. N. 1991 Finite State Approximations of Phrase Structure Grammars, in *Proceedings of the 29th Annual Meeting of the Association for Computational Linguistics*, ACL: Berkeley.

[1986] Pulman, S. G. 1986 Grammars, Parsers and Memory Limitations, in *Language and Cognitive Processes*, Vol 1, 3: pp. 197–225.

[1992] Ritchie, G. D., Black, A. W., Russell, G. J. and Pulman, S. G. 1992, *Computational Morphology*, MIT Press: Boston, Mass.