Variations on Incremental Interpretation

Stuart Shieber
Computer Science Department
Division of Applied Sciences
Harvard University
Cambridge, MA

Mark Johnson
Department of
Cognitive and Linguistic Sciences
Brown University
Providence, RI

January 22, 1993

Abstract

The strict competence hypothesis has sparked a small dialogue among several researchers attempting to understand its ramifications for human sentence processing and incremental interpretation in particular. In this paper, we review the dialogue, reconstructing the arguments in an attempt to make them more uniform and crisper, and provide our own analyses of certain of the issues that arise. We argue that strict competence, because it requires a synchronous computation mechanism, may actually lead to more complex, rather than simpler, models of incremental interpretation. Asynchronous computation, which is arguably both psychologically more plausible and conceptually more basic, allows for incremental interpretation to fall out naturally, without additional machinery for interpreting partial constituents. We show that this is true regardless of whether the presumed interpretation mechanism is top-down or bottom-up, contra previous conclusions in the literature, and propose a particular implementation of some of these ideas using a novel representation based on tree-adjoining grammars.

To appear in the *Journal of Psycholinguistic Research.*

Contents

1 Introduction

Mark Steedman (1989; 1992) argues that Occam's razor motivates a strict version of the competence hypothesis, under which the only structures that are interpreted by the human sentence processor are structures that are grammatically sanctioned. In particular, partial constituents are not interpreted under this hypothesis. As a consequence, the ability to understand utterances incrementally is evidence that combinatory categorial grammar (CCG) would seem to form the basis of a preferable theory of grammar, insofar as partial constituents are not needed to be interpreted under a CCG analysis.

This strict competence hypothesis has sparked a small dialogue among various researchers who are attempting to understand the exact relation among the various hypotheses in Steedman's argument: Occam's razor, strict competence, and incremental processing. In this paper, we review the dialogue, especially the contributions of Steedman, Stabler, and Abney, reconstructing the arguments in an attempt to make them more uniform and crisper, and provide our own analyses of certain of the issues that arise.

We argue that strict competence may actually be methodologically more complex a hypothesis, not simpler, as it implicitly assumes synchronous computation. Asynchronous computation, which is arguably both psychologically more plausible and conceptually more basic, allows for incremental interpretation to fall out naturally, without additional machinery for interpreting partial constituents. We show that this is true regardless of whether the presumed interpretation mechanism is top-down or bottom-up, contra previous conclusions in the literature, and propose a particular implementation of some of these ideas using a novel representation based on tree-adjoining grammars.

2 The Competence Hypothesis

> It is important to note that the strong competence hypothesis as stated by Bresnan and Kaplan imposes no further constraint on the processor. In particular, it does not limit the structures built by the processor to fully instantiated constituents. However, the present paper proposes a "strict" version of the competence hypothesis, which imposes this further condition. The reasoning behind this strict version is again evolutionary. If in order to process sentences we need more than the grammar itself, even a perfectly general "compiler" that turns grammars into algorithms dealing in other structures, then the load on evolution is increased. Similar arguments for the need for the grammar and processor to evolve in lockstep mean that a theory that keeps such extras to the minimum wins.
>
> This [strict] version of the competence hypothesis has the effect of

> generalising the constituent condition to cover the processor. The claim is that the constituents that are recognised in the grammar (and their interpretations) will be the only structures that the processor will give evidence of. Anything else whatsoever that we are forced to postulate is an extra assumption, and will require an independent explanation if it is not to count against the theory.
>
> (Steedman, 1992, page 37)

Steedman outlines several versions of the competence hypothesis, ranging in the restrictiveness of the hypothesis. The least restrictive variant, which might be called *weak competence*, requires that the output structures that the parser produces are those sanctioned by the competence grammar of the language. If the competence grammar is stated in terms of tree structures, the parser must output such tree structures. However, they may be computed as a result of a process that does not use the competence grammar, perhaps using some other grammar or set of tables to control the process.

Strong competence extends weak competence to require use of the competence grammar *per se* to control the processing of utterances. Nonetheless, the grammar may still be used to build structures that are not completed grammatical entities, for instance, partial tree structures.

The strongest form of the competence hypothesis, *strict competence*, disallows even this relatively benign use of partial structures. The structures that the parser uses (i.e., interprets) must be completed grammatical entities sanctioned by the competence grammar, not partial structures.

Strict competence ostensibly simplifies the syntax-semantics interface: the semantic interpreter in a system that does not respect strict competence must be able to interpret a larger set of constructions than in a system that obeys strict competence.

Steedman argues that in the face of other quite reasonable assumptions, strict competence provides evidence for a theory of grammar along the lines of combinatory categorial grammar. The argument is based on the following assumptions:

Modularity: The human sentence processor is divided into separate modules for syntactic processing and semantic processing. The structures generated by the syntactic processor are exactly those available to the semantic processor.

Semantic sensitivity: The syntactic processor is sensitive to aspects of the utterance that depend on the semantics of the syntactic structures being built. This sensitivity could take a wide variety of forms: The tightest kind of sensitivity would be detailed feedback based on semantic and pragmatic aspects of the utterance, as argued, for instance, by Altmann, Crain, Steedman, Tyler, Marslen-Wilson, and others (Altmann and Steedman, 1988; Crain and Steedman, 1985; Tyler and Marslen-Wilson, 1977).

At the other extreme would be a single reset or error signal to the syntactic processor to stop processing because of a semantic anomaly. This kind of loose interaction model is, for instance, the model of Frazier and others (Frazier, 1987).

Left-to-Right Incremental Interpretation: The syntactic processor operates from left-to-right, and to whatever extent it is sensitive to semantic effects, it is sensitive incrementally. Under a tight interaction model, for instance, semantic and pragmatic aspects of the utterance can disambiguate syntactic processing in an on-line manner. Under a loose interaction model, the reset signal can occur before the end of the sentence.

2.1 Incremental Interpretation and Disambiguation

These assumptions, taken together with the strict competence hypothesis, give rise to the following argument for a theory based on combinatory categorial grammar, as opposed to one based on more traditional structural analyses such as government-binding theory.

Consider the sentence

(1) The flowers sent for the patients arrived.

This sentence is locally syntactically ambiguous giving rise to the semantic distinction as to whether the flowers in question are the agent or the patient of the sending. Of course, the sentence is globally unambiguous; only the latter reading is possible. Crain and Steedman (1985) argue that disambiguation occurs at the word *sent*, at which time pragmatic information is available based on the semantic analysis of the locally ambiguous fragment *the flowers sent*. Under a loose interaction model, the hypothesis would be that disambiguation does not occur at that point, but rather, a reset signal is sent to the syntactic processor so that processing of the sentence can continue by virtue of some other type of sentence processing mechanism than the "automatic" system.

In order to determine that the NP *the flowers* is not a possible subject of the V *sent*, strict competence requires that there is a single constituent dominating the string *the flowers sent*, whose interpretation can be identified as anomalous. Moreover, by strong competence this constituent must be admitted by the grammar, but in traditional grammars the smallest such constituent is the node dominating the whole sentence. This constituent is not completed until the end of the sentence, and hence strict competence would predict that disambiguation cannot take place until the entire sentence has been processed, contra the assumption of incremental interpretation. Consequently traditional grammars do not provide structures that allow incremental interpretation under the strict competence hypothesis. Combinatory categorial grammars do provide such constituents; hence Steedman argues that CCGs are more plausible on psycholinguistic grounds than grammars admitting traditional structural analyses are.

3 Asynchronous versus Synchronous Computation

> No one seriously proposes pedestrian approaches to language understanding according to which *sentences* are fully parsed and then interpreted. So what could make the three assumptions with which we began seem paradoxical? It is just the idea that, although we do not need to have complete sentence structures before interpretation begins, we do at least need complete syntactic constituents, such as phrases. But why keep this assumption? The structures of phrases are complex (*i.e.*, composed of a number of elements), as are the structures of sentences, so we can perfectly well assume that the constituent parts of phrase structures become available for interpretation as soon as syntactic analysis formulates them. Furthermore, since the interpretation of a syntactic constituent is definitionally and procedurally complex, typically involving the interpretation of subconstituents, these subtasks can be intermingled with the parsing subtasks, *without making any modification at all in our definition of what each task involves.*
>
> (Stabler, 1991, page 203)

Stabler (1991) denies Steedman's conclusion on the grounds that the strict competence hypothesis is not needed. To make good this claim, he constructs example parser/interpreters (one operating top-down, the other bottom-up) using standard logic programming techniques to demonstrate that an incremental interpreter can be built that operates only over structures that are sanctioned by the competence grammar.

Nonetheless, Stabler does not explicitly address the logical rejoinder to his rhetorical question eschewing strict competence, "Why keep this assumption?" The reply would be that this assumption is simpler, and thus methodologically preferred. An interpreter of the Stabler type, it might be argued, must know how to interpret not only the structures that the competence grammar specifies as complete, but *other partial structures* as well. It is therefore a more complicated (and evolutionarily less likely) component of the human sentence processor. It is this rejoinder that we will directly address in this section centering on the distinction between synchronous and asynchronous computation, and we will review Stabler's top-down parser/interpreter construction in this light.

3.1 Models of Computation

Any computation from finite inputs to finite outputs can be thought of as a circuit made up of interconnected gates connecting the inputs to the outputs. (Presumably, one's brain can be thought of in this way as well.) For instance,

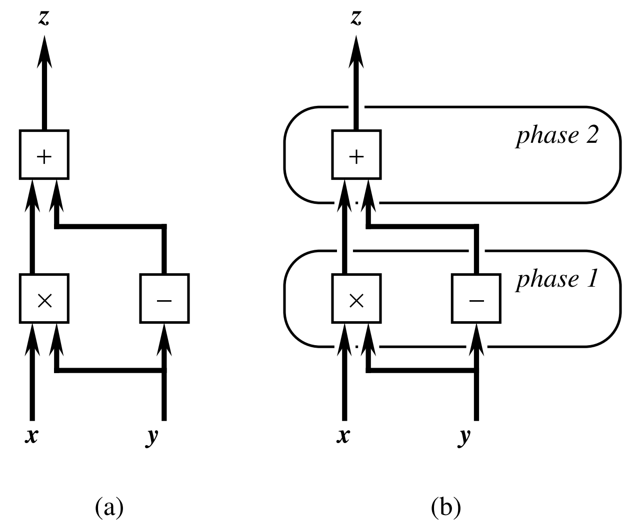

Figure 1: A circuit for computing $z = xy + (-y)$. (a) The circuit as a network of gates. (b) The circuit divided into two phases of computation.

the computation from inputs x and y to output z defined by the equation $z = xy + (-y)$ can be specified by the circuit of Figure 1(a).

There are at least two models of how such a circuit computes its output from its inputs. In the first model, *synchronous computation*, each gate in the circuit waits until all of its inputs are available before computing its output. In the second model, *asynchronous computation*, each gate computes its output as soon as it can, that is, as soon as its inputs are sufficiently specified to determine its output.

In fact, the level at which synchronization occurs in a circuit is not, as we have intimated, a binary choice. Above we described asynchronous computation, and computation synchronized at the gate level, but synchronization could occur at higher levels as well. Let us define certain groups of gates in the sample circuit as performing subcomputation. We will define the multiplication and negation gate as forming phase 1 of the computation and the addition as forming phase 2. Then we could have synchronization at the phase level of the circuit as well, that is, the output to phase 1 would only be generated when all of the inputs to phase 1 were available, and similarly for phase 2. (Since phase 2 has only one gate, gate-level and phase-level synchronization are the same for this phase.) Gate-level and phase-level synchronization are in general independent. Although we have specified synchronous computation at the phase level, we may still compute within each phase either synchronously or asynchronously at the gate level. Higher levels of synchronization can be imagined as well.

The import of the synchronous/asynchronous distinction can be exemplified with the sample circuit. Suppose we use the circuit synchronously at the phase level to compute z where y is 3 and x is 4. We set the y input to 3 by placing a 3 on the y wire. Although the negation gate has all of its inputs now, its output is not computed, since the outputs of the phase must wait synchronously on the availability of all the inputs. When x is then set to 4, all inputs for the first phase are available and the two outputs 12 and -3 can be generated. Finally, the second phase has all of its inputs and can compute its output, 9. (See Figure 2.)

In an asynchronous computation, on the other hand, as soon as y is set to 3, the negation gate can compute its output -3. But at this point computation must stop as no other gates have sufficient input. When x is set to 4, the remaining computation can proceed. (See Figure 3.)

The advantages of asynchronous computation can be seen when we examine the case where y is 0. The y wire is set to 0, allowing for the negation gate to set its output (phase-level asynchronously) to 0 as well. The multiplication gate, operating asynchronously at the gate level can set its output to 0, since a zero term in a product yields a zero result. The two inputs to the plus gate are thus set to 0, and the output can be computed as 0, even before the x gate is set to its value 4. (See Figure 4.) The synchronous computation would, of course, follow the same time-course behavior as in the case where y was 3.

Asynchronous circuits exhibit a kind of *eager* computation. In a sense, the circuit specifies, at a competence level, the computation of a function in terms of computational subconstituents. Yet the eagerness given by asynchronous computation allows the circuit to be used to compute values on partial inputs (for instance, where y alone is specified) with no change to the "competence" circuit at all.

Now, let us review the structure of the human sentence processor under the assumptions discussed in Section 2. The modularity assumption tells us that there are two phases in the circuit, one for syntactic processing and one for semantic interpretation. The inputs to the former is the sentence, and the strong competence hypothesis says that the computation uses the competence grammar directly, building as the output of the first phase complete grammatically-sanctioned trees. These serve as the input to the second phase of semantic processing. The output of this phase, because of semantic sensitivity and left-to-right incremental interpretation, must be available before the end of the sentence. In a tight interaction model, the output loops back to the first phase forming a recurrent circuit; in a loose interaction model, the output signals the second stage parser to take over the processing of the sentence from the "automatic" parser, and may also signal the first phase to reset. In either case, the circuit looks, schematically, as in Figure 5.

The strict competence assumption governing whether the semantic interpreter can use partial constituents developed by the syntactic processor, then, boils down to this: Humans use the circuit of Figure 5 in a *phase-level-synchronous* manner.

This way of viewing strict competence puts quite a different spin on judgments of simplicity. Just as asynchronous computation leads to incremental

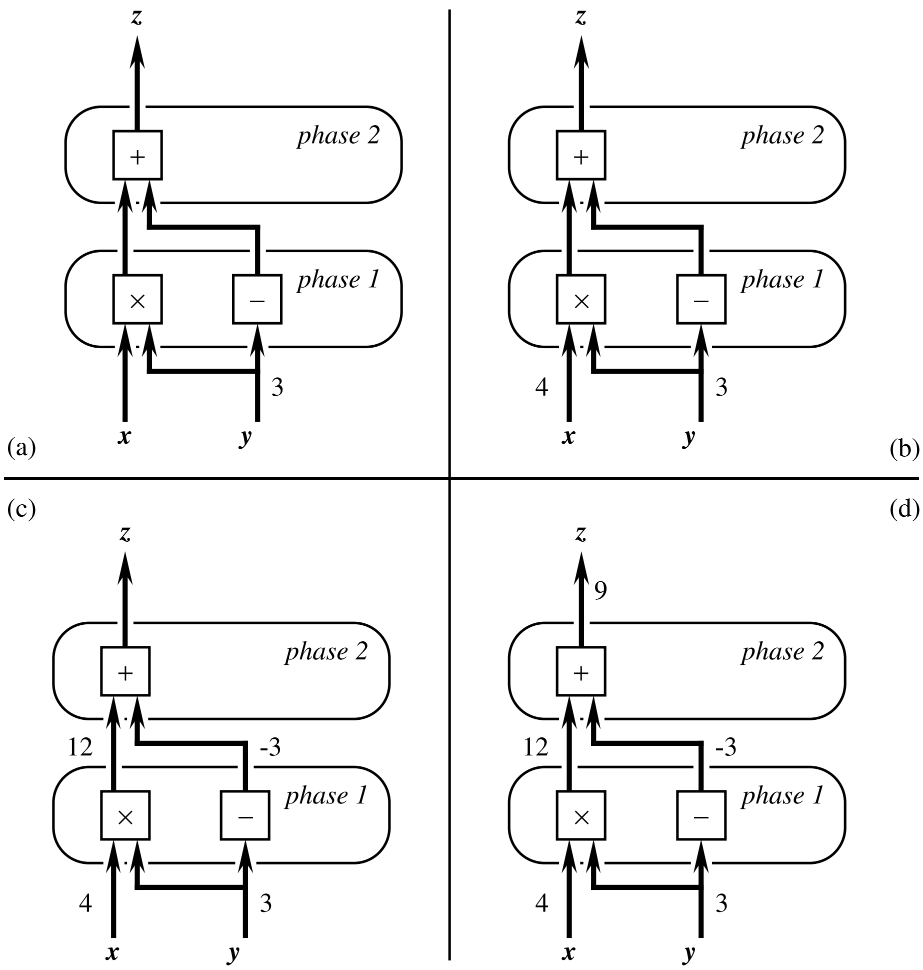

Figure 2: Phase-level synchronous computation of the sample circuit. (a) The y input is set to 3. Computation of the first phase must wait for the remaining input to the phase. (b) The x input is set to 4, allowing phase 1 computation to proceed. (c) Phase 1 computes its outputs, 12 and -3. (d) When phase 1 has completed computation, all inputs are available to phase 2. It computes the final output 9.

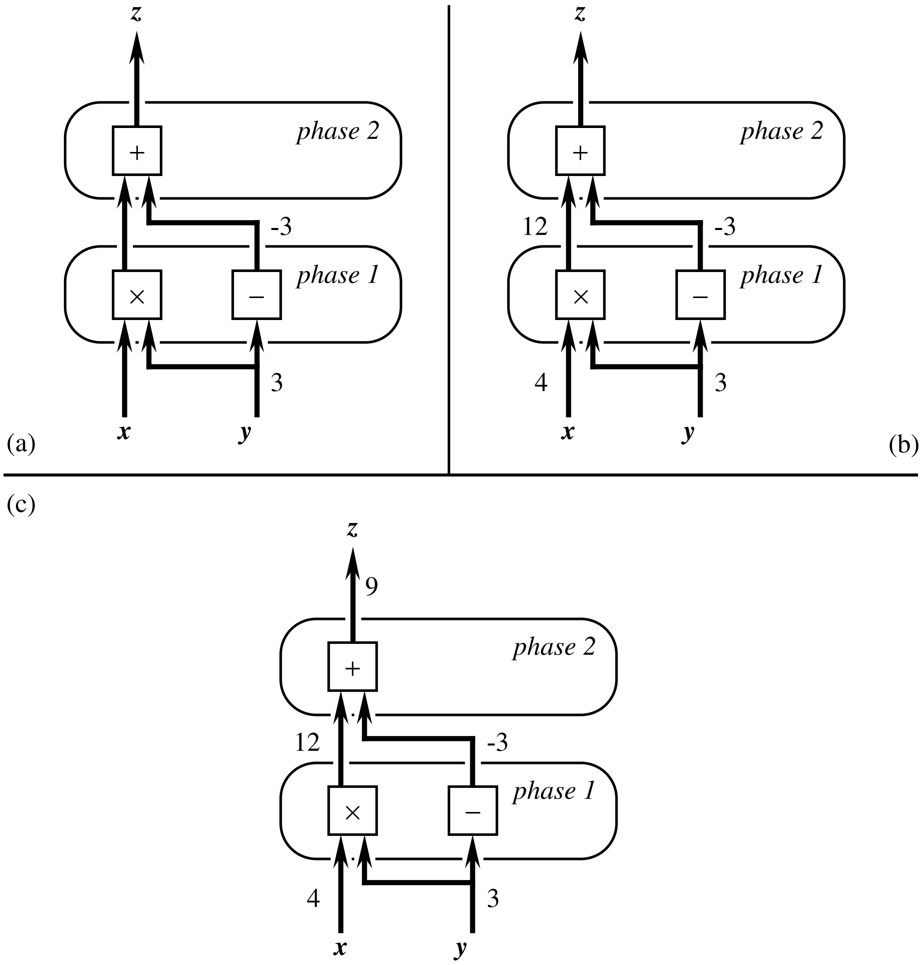

Figure 3: Asynchronous computation of the sample circuit. (a) The y input is set to 3. Since the negation gate has sufficient information to compute its output, its output is set to -3 immediately, but no further gates can operate. (b) The x input is set to 4. The multiplication gate can now compute its output 12. (c) The addition gate computes the final output 9.

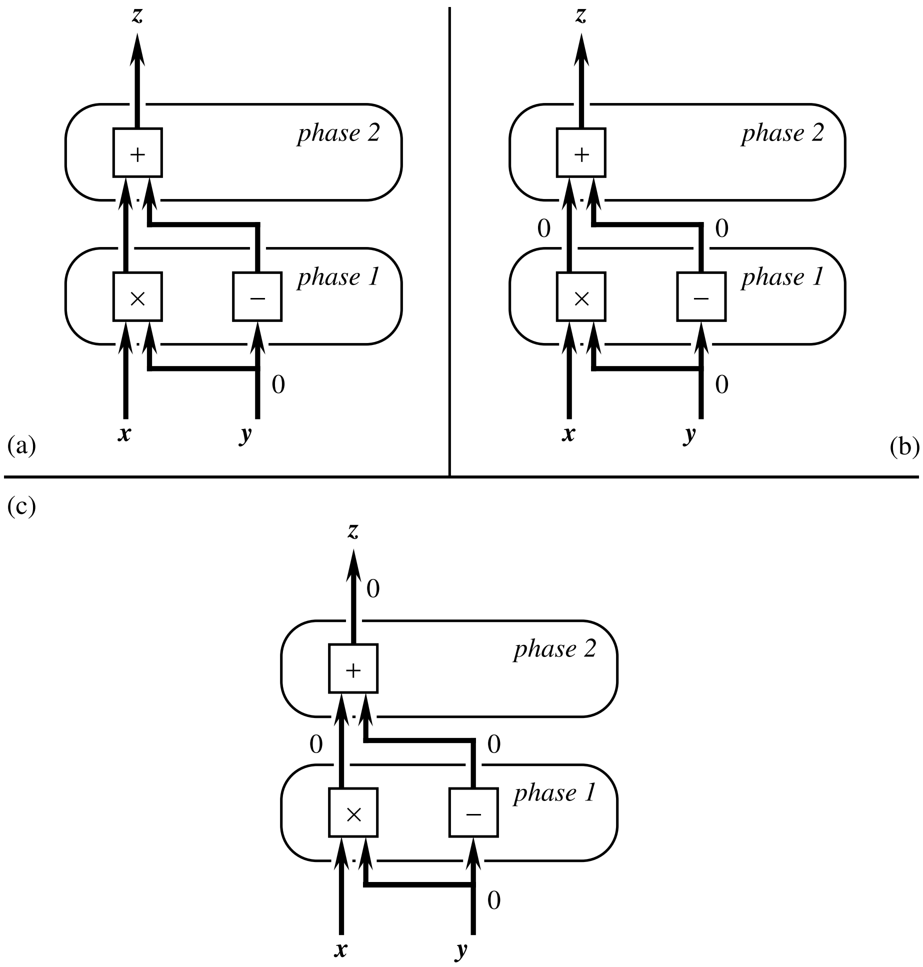

Figure 4: Asynchronous computation of the circuit showing eagerness of computation. (a) The y input is set to 0. (b) Both the phase 1 gates have sufficient information to compute their outputs, and asynchronously do so. (c) The addition gate has both inputs, and computes the final sum. Note that the final output 0 was computed even before the x input was provided.

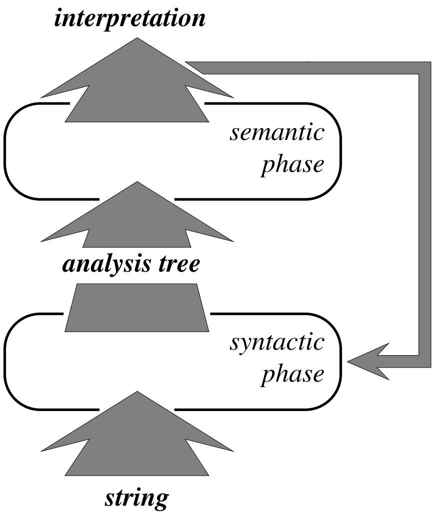

Figure 5: The model of human sentence processing viewed as a circuit. The input is a natural-language string. Two phases of computation are assumed (based on modularity). The first, a syntactic phase, computes (by strong competence) an analysis tree. The second, a semantic phase, interprets analysis trees, computing interpretations. Some aspects of the interpretation may be made available to the syntactic processor incrementally.

interpretation of partial results in the $z = xy + (-y)$ circuit without any extra work of explicitly specifying how to compute over partial structures, it can lead to incremental interpretation of natural-language utterances without any extra work of explicitly specifying how to interpret partial constituents. It seems then, that Steedman's claim that processors obeying strict competence are simpler is not necessarily true. In fact, as is familiar from circuit design, building a synchronous circuit is actually more complex than an asynchronous one. Synchronous circuits in general require global clocking mechanisms to prevent asynchronous computation from occurring. In a sense, asynchronous computation is a more primitive notion than synchronous computation. Thus, it might be argued that maintaining strict competence requires the extra infrastructure of synchronization, and is thus less simple rather than more.

Two issues remain open. First, we have argued that there is some reason to believe that strict competence is actually a more complex rather than simplifying assumption, but such an argument is not a proof. The human sentence processing circuit, the brain, may indeed be synchronous. In that case, strict competence may be a simplifying assumption after all. We know of no way at the present to resolve this issue, so it must remain open. Second, we have claimed that asynchronous computation can allow for incremental interpretation without any extra work of specifying how to interpret partial constituents. This claim has been argued for in detail by Stabler. We will review his construction in the next section.

3.2 Asynchronous Computation and Connectionist Networks

We digress briefly to note that a circuit model such as we have outlined bears an obvious resemblance to connectionist networks. The feedback in the circuit makes recurrent connectionist networks (Elman, 1988) an obvious candidate for subsymbolic implementation of the ideas presented in this paper.

Although Elman's version of recurrent connectionist networks has assumed synchronous updating of activation levels, many researchers have proposed asynchronous updating as a preferable method for both technical and psychological reasons. Hopfield networks (Hopfield, 1982) are an early example of the asynchronous connectionist approach. More recently, Smolensky (1986) has proposed an asynchronous model called *harmonium* and applied it to various natural-language areas (Legendre, Miyata, and Smolensky, 1990). In fact, the synchronous update method "is usually viewed as a discrete, difference approximation to an underlying continuous, differential equation in which all units are continuously [i.e., asynchronously] updated." (Rumelhart, Hinton, and McClelland, 1986, page 61)

Asynchronous computation is thus a natural property of connectionist systems; insofar as connectionist models are plausible models of human mental processes, then asynchronous computation would be an expected property of human sentence processing.

4 Asynchronous Computation and Incremental Interpretation

Stabler makes the general point that syntax, semantics, and truth computation can be interleaved in a general asynchronous deduction architecture. As an example, he presents a parser that generates analysis trees in a top-down fashion by recursive descent. The output of this parser serves immediately as input to an interpretation mechanism, which, when operating asynchronously (as it does in Stabler's deduction architecture) gives rise to incremental interpretation.

4.1 Top-Down Incremental Interpretation

We will not repeat the programming details here, referring the reader instead to Stabler's original paper (1991). However, we will go through an informal derivation (shown in Figure 6) to give a flavor of the idea. Parsing proceeds top-down from the start symbol of the grammar, S. This symbol is expanded according to the grammar, as in Figure 6(a), to an *NP* followed by a *VP*. The corresponding semantic operation, application of the *VP* meaning to the *NP* meaning, can be asynchronously generated top-down as well. (We notate semantic interpretations not using a standard logical notation, e.g., $sent'(x)(the'(flowers'))$ for the sentence prefix *the flowers sent*, but rather as a tree structure in which the applications of functors to their arguments are made explicit with the label *ap*. This detail is further discussed in Section 4.2.) In step (b), the leftmost unexpanded nonterminal (*NP*) is expanded, and the corresponding semantic operation is asynchronously added to the semantic representation. Again, the leftmost nonterminal is expanded, this time to a terminal *the*, and the semantic contribution is incorporated into the semantic representation, as shown in (c). Continuing in this way, the syntactic tree is traversed top-down, depth-first, and left-to-right, while the semantic representation is built incrementally according to a similar traversal. In the final step shown (f), the string *the flowers sent* has been parsed. Already, the semantic representation records that the interpretation of *the flowers* serves as the second argument (the agent argument in the curried representation) to the predicate $sent'$.

Given that this derivation is pursued top-down, at the point at which *the flowers sent* has been parsed, the interpreter has already constructed an interpretation placing the flowers as the agent of the sending. Thus the implausibility of this reading is available well before the end of the sentence, and indeed, at the point where the disambiguation actually occurs.

Unfortunately, the top-down parser was confronted with this ambiguity much earlier. The choice to expand the *NP* top-down as a *Det* followed by an *N* (as opposed to, say, an *NP* followed by a reduced relative clause) essentially commits the top-down parser to this reading, and that choice occurs at the beginning of the constituent, i.e., the beginning of the sentence. In general, a top-down parser makes such commitments too early.

Nonetheless, the example shows that asynchronous computation can directly

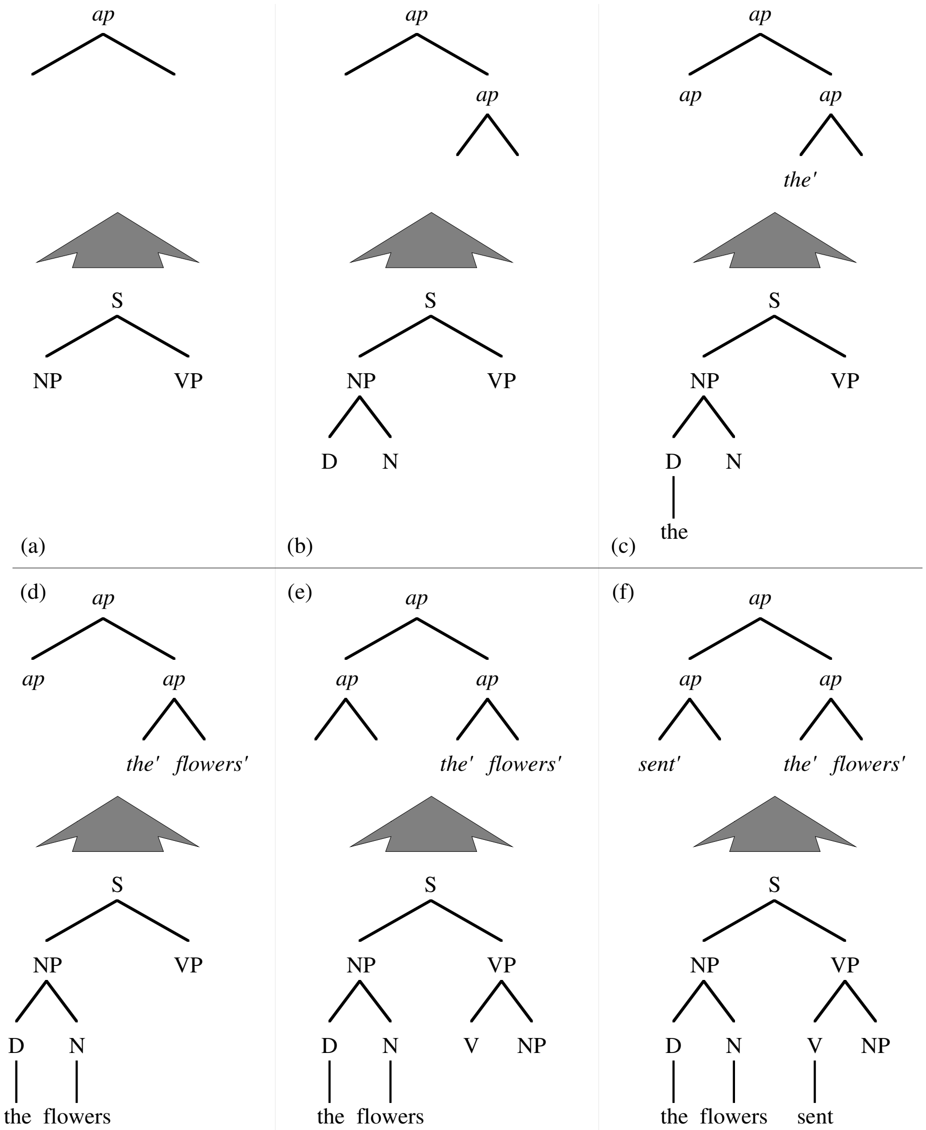

Figure 6: The first several steps of a top-down parse generating an analysis tree in a top-down manner. At each step, the interpreter can asynchronously compute a semantic representation top-down as well.

give incremental interpretation, as we were wont to show, even if it cannot articulate that incremental interpretation with a disambiguation component. To get incremental interpretation to work with disambiguation requires postponement of choice points; a bottom-up parser is a natural way to achieve this postponement.

4.2 Bottom-Up Incremental Interpretation

> ...LR parsers cannot model the incrementality of human parsing. Namely, in right-branching structures, LR parsers build no structure until all the input has been read. ...By contrast, people clearly build structure before the end of the sentence and pass it incrementally to the semantic processor.
>
> (Abney, 1989, page 130)

In order to show that incremental interpretation can not only be done but done in a useful manner, so that the interpretations are available where the local choice points occur, we would like to apply the same kind of techniques based on asynchronous computation to the building of a parser with later choice points. In this section we consider a quite extreme case of postponing choice points, bottom-up parsing. Bottom-up parsers do not commit to choices as to the structure of a constituent until the point in the string that corresponds to the end of the constituent. It should become obvious from the later discussion that if an incremental interpretation mechanism can be constructed naturally through asynchronous computation for this case, then one could be constructed for parsers that commit earlier than a bottom-up parser (but still later than a top-down one), such as left-corner parsers. It is for this reason that we choose to work on the most difficult case.

Abney (1989) argues that although Stabler's method, which we have characterized as asynchronous computation, works for using top-down parsers for incremental interpretation, it necessarily fails for bottom-up parsers. (He considers the particular case of LR parsers.) By way of review, an LR parser is a kind of automaton that has a current state (one of a finite set of states) and a stack of constituents that have been parsed so far. Each step that the LR automaton makes falls into one of four classes: The automaton may *shift* the next word onto the top of the stack. The automaton may *reduce* the top several constituents on the stack by replacing them with a single new constituent that consists of a new node dominating the constituents that were previously on top of the stack. The automaton may *accept* the string as being grammatical and halt. The automaton may *reject* the string as being ungrammatical and halt. We will be most concerned with the first two of these operations. As an example, the LR parse of sentence (1) is given in Figure 7.

Abney points out that the stack of categories in an LR parser does not appear to provide enough committed structure for incremental interpretation. In our

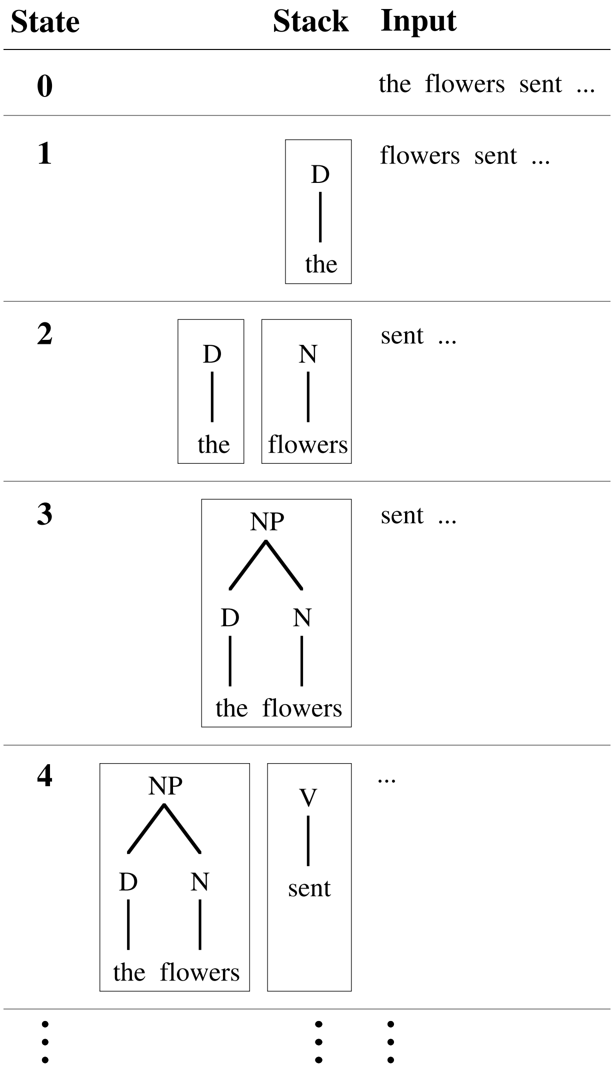

Figure 7: The first few steps of a shift-reduce parse of a string beginning "the flowers sent...".

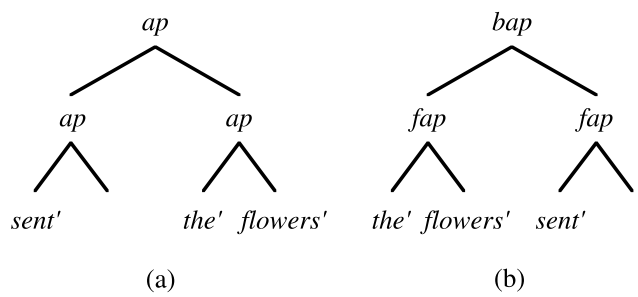

Figure 8: Two depictions of a semantic representation. (a) Depiction of the logical form $sent'(x)(the'(flowers'))$ as an applicative structure, with applications explicitly marked as ap. (b) A directed version with applications marked bap for backward application and fap for forward application.

example, the relationship between the NP *the flowers* and the V *sent* is not specified in the LR parser's stack until at least the end of the VP dominating *sent* has been reached. But this is the end of the sentence. Thus, we are back to the old problem of having to wait until the end of the sentence to do interpretation.

> ...it is easy to see that this problem is similarly an artifact of inessential formal details.... Given the new [bottom-up] notation for our syntactic structures, it is natural to use a bottom-up notation for our LF representations as well. Instead of using the form
>
> $$sat(joke(0), funny)$$
>
> let's use
>
> $$[joke, 0, INP/ICP, funny, sat(INP, IVP)] \quad .$$
>
> (Stabler, 1991, page 218-219)

Stabler responded that LR parsers can produce what we will call *postfix applicative structure*. These structures are a postfix variant of the applicative semantic structures that we have been using. We have been notating semantic interpretations using a standard logical notation, e.g., $sent'(x)(the'(flowers'))$ for the sentence prefix *the flowers sent*. Such a notation can be presented more explicitly as a tree structure, as in Figure 8(a), in which the applications of functors to their arguments is made explicit with the label ap. We will call such logical notations *applicative semantic structures.* A variant of applicative semantic structures, in which we distinguish a forward from a backward variant of application (fap and bap, respectively), can also be developed, at the expense of allowing some redundancy in notation. A directed applicative semantic structure is given in Figure 8(b). Just as a prefix traversal of the applicative semantic structure yields the equivalent logical notation (i.e., $sent'(x)(the'(flowers'))$), a postfix traversal of the directed applicative structure yields another notation for the interpretation

$$[the', flowers', fap, sent', \ldots, bap]$$

that is, in some sense, equivalent to the other notations. We will call this *postfix applicative structure.* Stabler's point is, essentially, that the postfix applicative structure for a sentence carries just the same information as the prefix version, but is generable incrementally by an LR parser. In fact, Stabler presents a second parser/interpreter, again an instance of an asynchronous interleaving of two standard phases, which performs just this incremental generation of postfix applicative structure. This would seem to answer Abney's qualms.

Unfortunately, we do not believe that it does. The problem is a bit deeper than being "an artifact of inessential details" as to whether a prefix or postfix notation is used for representing semantics. Postfix applicative structures are essentially lists of logical elements. The subcomponents of such a list, as evidenced by the way in which Stabler's system builds them up incrementally are the prefixes of the list. Thus, the postfix applicative structure for sentence (1) would be built up incrementally as

$$[the']$$
$$[the', flowers']$$
$$[the', flowers', fap]$$
$$[the', flowers', fap, sent']$$
$$\ldots$$

But these subcomponents are not compositionally interpretable in a way sufficient for disambiguation. For example, after the prefix *the flowers sent*, in Stabler's system the incremental semantics consists of the postfix applicative structure $[the', flowers', fap, sent']$, which is not compositionally interpretable in such a way that it manifests the relation between the flowers and the sending. The implementation of an incremental interpretation system that Stabler provides is a top-down one that not only incrementally develops semantic representations but also evalautes them to determine their truth relative to a model. The ability to use the semantic representations as an appropriate input to an evaluator is central. Significantly, Stabler provides no method for using postfix applicative structures as input to an incremental evaluator. This is a symptom of the underlying problem with bottom-up incremental interpretation.

Stabler's method therefore does not solve the problem of incremental interpretation with LR parsers; it postpones the problem. This is not surprising, as it is actually the case that the stack *does not* have sufficient information to make the connection between the flowers and the sending at that point in the sentence. Any method, like Stabler's, that essentially builds a semantic representation based solely on the contents of the stack will fall foul of the ability to do disambiguation on the basis of that semantic representation.

Nonetheless, we argue that Abney's claim is wrong, though not for the reason that Stabler gives. Rather, what Abney has ignored is the fact that the configuration of an LR parser consists of more than just its stack contents. There is also a finite amount of state information. As it turns out, the state of an LR parser finitely encodes a set of possible left contexts for the stack items. This set of contexts has a regular structure, and corresponding to that regular structure of syntactic left contexts, there is a regular structure of functors over the completed constituent meanings (from the stack). Since these functors are incrementally computable from the LR state, they are accessible by the interpreter, hence available for incremental interpretation.

Again, we will present informally an LR derivation along with an asynchronous computation of the semantics. (See Figure 9.) After the first parsing step, the shifting of the word *the* entering state 1 (recalling Figure 7), the stack contains information about the single constituent, the determiner *the*. State 1 encodes an equivalence class of left contexts, representable graphically by the unboxed syntactic tree structure in Figure 9(a). The subtree notated with dotted lines is intended to indicate zero or more repetitions of the substructure. (The notation serves as a kind of Kleene star for tree structures.) Thus, the left contexts represented include all those with S dominating NP and VP, the NP perhaps dominating more NPs along the left branch, with the lowest NP dominating the determiner on the stack and an N. It is important to note that the parser does not build one or more of such contexts, or a tree-like representation of all of them (as drawn in the figure). The parser merely keeps a single token representing the state, 1, that can be extrinsically interpreted (by us or by the interpreter) as standing for these left contexts. Thus, the graphical depictions in the figure are mnemonic only, to make it easier to understand what information is available for processing, but not how it is represented.

The parser makes available to the interpreter the information written below the arrow in Figure 7(a). (Again, we emphasize that it need not be and is not made available *in this form*.) On the basis of this information, the interpreter can conclude that the semantic structure of the sentence is as represented in the tree structure above the arrow. Of course, as before, the interpreter need not manifest this information in this form. The figure merely shows what information is available. As the parsing steps proceed, more information is added to the parser output through the stack contents (always boxed in the figures) and finite state (unboxed). After the second shift, the situation is as depicted in Figure 9(b). The subsequent reduction step, shown in 9(c), does not change the information content, but reapportions it between the stack and state. The next shift (9(d)) adds further information both syntactically and semantically. The semantic information available to the interpreter at this point includes the

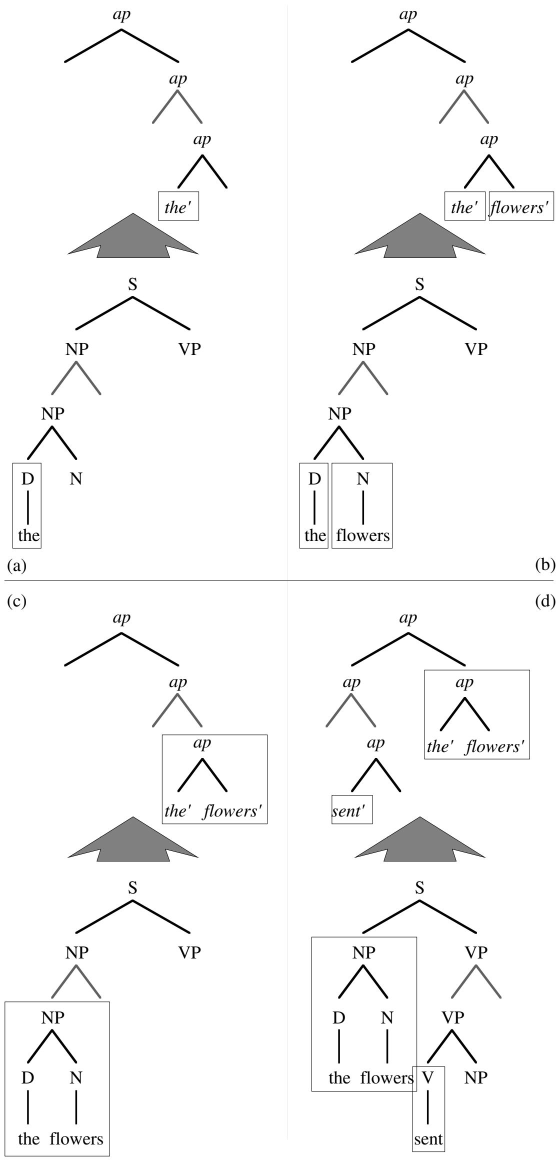

Figure 9: The first several steps of a bottom-up shift-reduce parse generating an analysis tree in a bottom-up manner. At each step, the interpreter can asynchronously compute a semantic representation bottom-up as well. The boxed material is the stack elements in the parser and the interpretations of the stack elements in the interpreter. The unboxed material represents the equivalence classes of left contexts that the LR state encodes and the interpretation of the state.

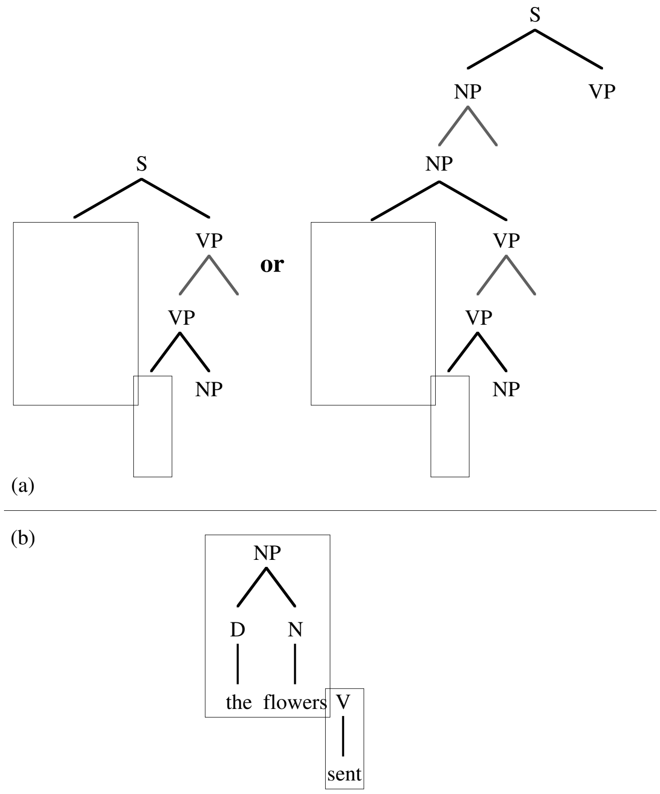

Figure 10: Syntactic information represented by the state (unboxed) and stack (boxed) of an LR parser after parsing of "the flowers sent". Note the disjunctive left-context information.

fact that $the'(flowers')$ is the agent of $sent'$; thus, the interpreter has sufficient information to perform disambiguation or to signal a reset at this point.

Actually, the derivation given in Figure 9 has been simplified. The left-context information represented by the states is more complex than that represented graphically in the figure. In particular, the left context information represented by state 4, the state after the shifting of the word *sent*, is not as shown in Figure 9(d), but rather as shown in Figure 10(a). It is, in fact, the disjunction of two sets of left contexts, each representable by a tree with Kleene-starred subtrees. (Recall that regular sets allow both Kleene star and disjunction. Thus, the left-context set is still regular.) As before, the syntactic information is as represented in Figure 10(b). Corresponding to this enlarged syntactic context that the state 3 represents, there is an enlarged semantic context as well that is available to the interpreter. This is shown in Figure 11(a). Essentially, the left

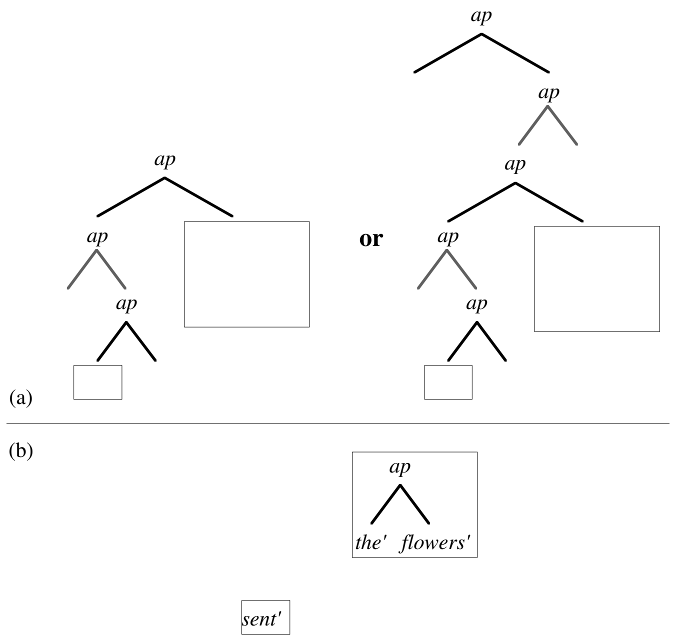

Figure 11: Semantic information represented by the state (unboxed) and stack (boxed) of an LR parser after parsing of "the flowers sent". Note the disjunctive left-context information.

disjunct corresponds to the interpretation of the sentence under which the flowers are the agent of the sending, the right disjunct to the interpretation where the flowers are the patient. This full presentation of the semantic information after the word *sent* shows that like the top-down case, sufficient information is available for disambiguation or anomaly detection, but furthermore, and unlike the top-down case, the disambiguation can be performed at this point. The choice has been postponed sufficiently long. Thus, this model allows both for incremental interpretation and appropriate articulation of the incremental interpretation with disambiguation.

5 Representation of Grammatical Constructs for Asynchronous Processing

Previous sections of this paper argue that all other things being equal, the null hypothesis concerning syntactic and semantic processing is that they should proceed asynchronously. Furthermore, we have shown that asynchronous processing can in theory allow for incremental interpretation regardless of whether language processing proceeds top-down, bottom-up or some combination of the two. The argument with respect to bottom-up processing was an in principle one; the information necessary for incremental interpretation is available, hidden in the stack and state, but no particulars were presented as to how the processing might go in a particular implementation. In this section, we discuss one way (presumably many others exist) that the ideas presented in the previous section can be fleshed out. In particular, we address the issue of "regular contexts", contexts that are partially indeterminate as to structure, which must be at least implicitly manifested. We show how the idea can be implemented using a recent tool from the computational linguistics literature, Synchronous Tree-Adjoining Grammars. We present a simple example involving VP adjunction — recall that this leads to a Kleene-star type of context — in which asynchronous processing has a certain intuitive appeal, and sketch a system in which semantic processing can in fact proceed beyond syntactic processing. Along the way certain issues of modularity in processing are raised and discussed.

If we consider syntactic processing as a process incrementally instantiating a parse tree for the portion of the utterance, then at the point in the parse marked by $\diamond$ in sentence (2) below, there is not sufficient information available to determine the number of VP nodes that will appear in the syntactic parse tree, because the number of post-verbal adjuncts cannot be determined from the portion of the utterance seen by the processor.

(2) George hates $\diamond$ broccoli violently.

At this stage, the precise tree-structural relationship— in particular, the number of intervening VP nodes—between the subject NP *George* and the verb *hates* cannot be determined, even though the subject-verb relationship between these elements can be.

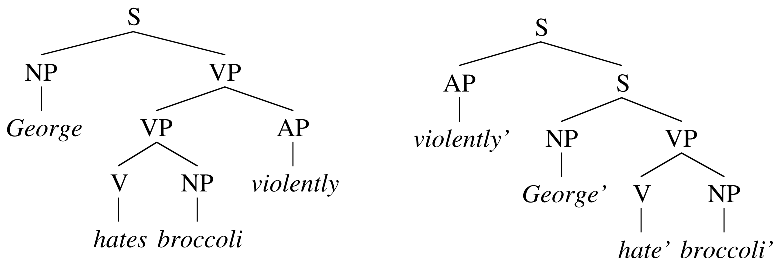

Figure 12: Adjunction to VP at s-structure and corresponding adjunction to S at LF

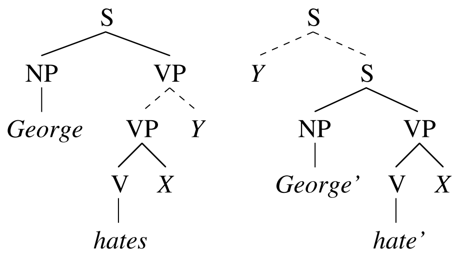

Figure 13: "Partial trees" constructed at the word *hates*

Suppose we adopt an LF-style of semantic representation (May, 1985) in which VP adjuncts are raised and adjoined to S (their "scope positions") at LF. Then the uncertainty as to the number of adjunctions to VP in the syntax corresponds to an uncertainty in the number of adjunctions to S (rather than VP) at LF, as sketched in Figure 12. At LF the grammatical relationship between *George* and *hates* can be read off of a single contiguous partial tree. In principle, then, an asynchronous semantic processor working at the level of LF could detect this grammatical relationship, check it for semantic and pragmatic plausibility, and issue instructions to the syntactic processor if needed.

Immediately after the verb *hates* the parser's information state about the syntactic and semantic parse trees might be informally characterized by the two trees sketched in Figure 13. In this diagram, the variable Y should be interpreted as a kind of "sequence variable", ranging over zero or more adjunctions, representing the fact that the number of adjunctions is undetermined.

Of course, these crude sketches do not demonstrate that there is in fact a systematic way to asynchronously construct syntactic and semantic representations. The remainder of this section demonstrates that there is at least one way to do this that has the property that semantic interpretation can proceed *before* the syntactic structure has been completely identified.

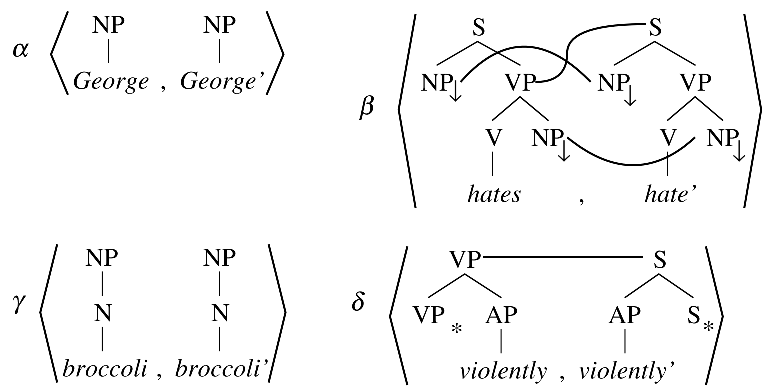

Figure 14: Paired syntactic-semantic trees used in the example

In the framework of Synchronous Tree-Adjoining Grammars[1] (STAG) as described by Shieber and Schabes (1990; 1991), the relationship between syntactic and semantic representations is expressed by means of linked pairs of *elementary* syntactic and semantic trees. Elementary trees come in two kinds: initial and auxiliary trees. Figure 14 depicts the paired elementary syntactic and semantic trees used in this example. The links between these pairs of trees synchronize the substitution and adjunction operations on the paired syntactic and semantic parse trees. In that figure, each pair is labeled with a greek letter, which will be used to refer to the pair in the text. The trees labeled α, β and γ are *initial* tree pairs. These trees can begin a derivation or be substituted for leaf nonterminals of other trees. The δ labels an *auxiliary* tree pair, one that can be adjoined into other trees.

A derivation begins by selecting an initial tree pair with a syntactic tree whose root is labeled S. In this example, the only such tree pair is β. The derivation proceeds by selecting one of the links and an initial or auxiliary tree pair and simultaneously substituting or adjoining (depending on whether the trees are initial or auxiliary) the trees at the two ends of the link, and deleting the link. (For the full technical details, see the papers by Shieber and Schabes cited above.)

For example, substituting α into the linked subject NPs of β and then substituting γ into the linked object NPs yields the pair of trees shown in Figure 15.

The LF raising of adjuncts is captured by the auxiliary tree pair δ. Because the VP node in the syntactic tree is linked with the S node in the semantic tree, a VP adjunction operation on the syntactic tree can only occur when a

[1]The term *synchronous* in the name of the formalism refers to the synchronization of syntactic and semantic specifications, not to synchronous processing. Indeed, it is the synchronization of the specification that allows asynchronous processing to be a viable processing strategy for STAGs.

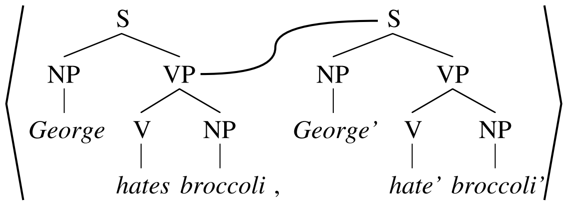

Figure 15: The result of substituting α and γ into β

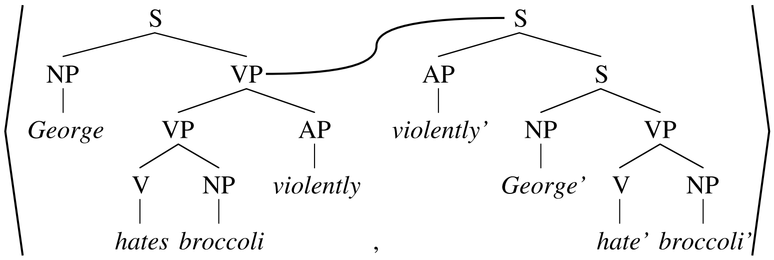

Figure 16: The result of adjoining δ into the tree pair depicted in Figure 15

corresponding S adjunction occurs in the semantic tree. The result of adjoining δ's syntactic tree at VP and δ's semantic tree at S is depicted in Figure 16.

Note that the link between the matrix VP and S nodes in δ also appears in the derived tree pair depicted in Figure 16, so an unbounded number of VP—S adjunctions is possible.

This concludes the brief description of synchronous tree-adjoining grammars. We, now turn to an abstract model of incremental interpretation based on the method proposed by Lang (1991) in the analysis of general parsing algorithms. Again, our treatment is informal; the reader should turn to the original paper for the details of the procedure.

The basic idea is very simple. Given a grammar G and a string of words w, consider the set S of all the trees (or, in the case of STAGs, tree pairs) generated by G whose yield begins with w. S is the set of all grammatical trees consistent with the input seen so far. In general S will be an infinite set, so S cannot be finitely characterized by simply enumerating its members.

But as Lang (1991) realized, in many cases S can be finitely characterized by a grammar G' that strongly generates a set of trees structurally isomorphic to S. Specifically, if G is a member of a class of grammars that is closed under intersection with finite-state languages (these include CFGs, TAGs, and STAGs), it must be the case that there is a grammar G' such that G' generates isomorphic versions of the trees in S. This G' is a finite description of the set

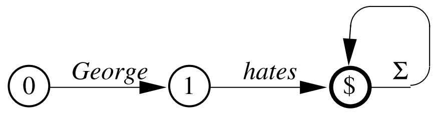

Figure 17: A finite-state machine M that accepts all strings with the prefix *George hates*

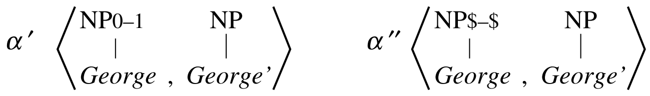

Figure 18: Specialized versions of the tree pair α from Figure 14

of all trees compatible with both the grammar and the input seen so far. In the case of STAGs, G' is a transduction between syntactic and semantic trees that are compatible with the input seen so far. We now sketch how G' can be calculated, and show how given the partial input *George hates* ... the NP *George* can be unambiguously identified as the subject of the verb *hates* at LF.

First, we construct a finite state machine M which accepts *George hates* Σ^*, i.e., all strings that begin with *George hates*. Such a machine is depicted in Figure 17. In this machine the state labeled 0 is the start state, and the state labeled \$ is a final, accepting state.

Next, the tree pairs in the original grammar depicted in Figure 14 are specialized. In the specialized grammar each nonterminal in a syntactic tree is annotated with a tuple of states from the finite state machine M that indicate which substrings of the language of M that it may dominate. Nonterminals that are not potential adjunction sites are annotated with a pair of states, corresponding to the left and right string positions of the substring they dominate, while nonterminals that are potential adjunction sites are annotated with a quadruple of states, corresponding to the upper and lower pairs of string positions of the node after a potential adjunction.

For example, the syntactic tree in the pair α of Figure 14 can span strings of the state pairs $0 - 1$ and $\$ - \$$, so that tree is specialized into the two trees α' and α'' shown in Figure 18.

On the other hand, the string *broccoli* can only possibly occur in the Σ-loop part of M, so the tree pair γ has only the single specialization depicted in Figure 19.

Now consider initial tree β. Its root node (the S node) must span the entire string, so its specialization must be annotated $0 - \$$, i.e., the initial and final states. Since the S node's left-most child (the subject NP node) must have the same left string position as the S node, the first component of its annotation

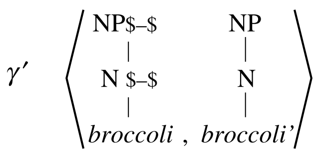

Figure 19: The specialized version of the tree pair γ from Figure 14

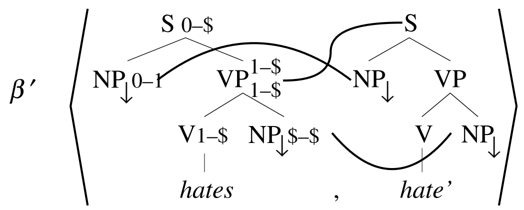

Figure 20: The specialized version of the tree pair β from Figure 14

must be 0. But there is only one syntactic tree with a root NP node at left string position 0, namely the tree in the pair α'. Since α''s right string position is 1, the right string position of the subject NP node in any specialization of β must be 1 also.

By similar reasoning we can determine the left string positions of the VP and V nodes in any specialization of β, and from that we can fix their right string positions as well. Thus the specialization of β is completely determined; it is given in Figure 20.

Finally, consider the specialization of the auxiliary tree δ. Clearly, only those specializations labeled $\mathrm{VP}^{1-\$}_{1-\$}$ are potentially useful in the specialized grammar G', since specializations with other labels will not be able to adjoin to the tree pair β'. There is only one such specialization, which is given in Figure 21.

The specialized grammar G' thus consists of the tree pairs α', α'', β', γ' and δ'. It defines the syntactic-semantic transduction restricted to syntactic trees whose yields have the prefix *George hates*. There are an infinite number of tree pairs in this transduction, since the tree pair δ' can be recursively adjoined.

Now consider the semantic trees that can appear as the output of the transduction defined by G'. Since α' is the only tree pair that can substitute into the subject NP in β', the NP *George* must appear in this position. Further, even though the tree pair δ' can be recursively adjoined, at the semantic level this adjunction is an adjunction to S, not to VP. Thus all of the semantic trees defined by G' are of the general form sketched in Figure 22.

Thus, even though the exact sequence of nodes intervening between the subject NP and the verb in the syntactic tree cannot be determined with only

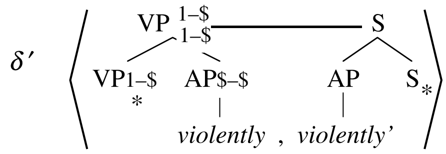

Figure 21: The specialized version of the tree pair β from Figure 14

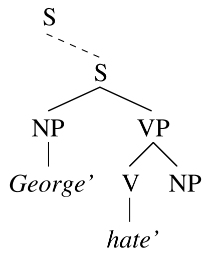

Figure 22: The general form of the semantic trees generated by the specialized grammar

the first two words of the input, at the semantic level this sequence of intervening nodes is fully determinate, and so the grammatical relationship at LF between the verb and the subject NP can be unambiguously identified.

In essence, synchronous tree-adjoining grammars provide a framework in which regular contexts of the variety needed for bottom-up incremental interpretation can be naturally found. The ability to perform adjunctions in the middle of derived trees allows for a single tree to stand proxy for an entire set of contexts, and the synchronization with a semantic representation allows for the asynchronous development of the semantic relationships simultaneously with the syntactic ones.

6 Conclusion

We have argued that the strict competence hypothesis proposed by Steedman may not, as was previously proposed, constitute a simplifying assumption for models of natural-language interpretation. The naturalness of asynchronous computation, and its ability to manifest incremental interpretation without any additional machinery beyond that required for a nonincremental interpretation process, argues for relaxing the strict competence assumption. Indeed, once the strict competence assumption is seen as a requirement of synchronous processing, it is apparent that the assumption complicates language interpretation models by requiring a clocking or synchronization mechanism that is otherwise unnecessary.

With the strict competence assumption relaxed, we have argued (with Stabler) that incremental interpretation can naturally take place by asynchronous computation of semantic representations; this follows independently of whether processing occurs top-down or (contra Abney) bottom-up. A concrete version of the bottom-up proposal might use synchronous tree-adjoining grammars to handle the representation both of grammar and of incremental results. Along these lines, the use of tree-adjoining grammars as an appropriate formalism for stating the observations of modern linguistic theory (Kroch and Joshi, 1985; Kroch, 1989; Frank, 1992) are pertinent.

Although the proposals we outline in this paper are programmatic, the general conclusion is, we believe, reasonable: Incremental interpretation follows from asynchronous processing directly, without the necessity to hypothesize additional assumptions.

Acknowledgments

The research in this paper was supported in part by grant IRI-9157996 from the National Science Foundation to the first author.

The authors would like to thank Fernando Pereira, Edward Stabler, and Mark Steedman for discussions on the topic of this paper and for their comments on previous drafts.

References

Abney, Steven P. 1989. A computational model of human parsing. *Journal of Psycholinguistic Research*, 18(1):129–144.

Altmann, Gerry and Mark Steedman. 1988. Interaction with the context in human syntactic processing. *Cognition*, 30:191–238.

Crain, Stephen and Mark Steedman. 1985. On not being led up the garden path: The use of context by the psychological parser. In David Dowty, Lauri Karttunen, and Arnold Zwicky, editors, *Natural Language Parsing: Psychological, Computational, and Theoretical Perspectives.* Cambridge University Pres, Cambridge, England.

Elman, Jeffrey L. 1988. Finding structure in time. Technical Report 8801, Center for Research in Language, University of California, San Diego, La Jolla, California, April.

Frank, Robert. 1992. *Syntactic Locality and Tree Adjoining Grammar: Grammatical, Acquisition and Processing Perspectives.* Ph.D. thesis, University of Pennsylvania.

Frazier, Lyn. 1987. Theories of sentence processing. In Jay L. Garfield, editor, *Modularity in Knowledge Representation and Natural-Language Understanding*. MIT Press, Cambridge, Massachusetts, pages 291–307.

Hopfield, J. J. 1982. Neural networks and physical systems with emergent collective computational abilities. *Proceedings of the National Academy of Sciences*, 79:2554–2558.

Kroch, Anthony. 1989. Asymmetries in long distance extraction in a tag grammar. In M. Baltin and A. Kroch, editors, *Alternative Conceptions of Phrase Structure*. University of Chicago Press, pages 66–98.

Kroch, Anthony and Aravind K. Joshi. 1985. Linguistic relevance of tree adjoining grammars. Technical Report MS-CIS-85-18, Department of Computer and Information Science, University of Pennsylvania, April.

Lang, Bernard. 1991. Towards a uniform formal framework for parsing. In Masaru Tomita, editor, *Current Issues in Parsing Technology*. Kluwer Academic Publishers, Dordrecht, Holland, pages 153–171.

Legendre, Géraldine, Yoshiro Miyata, and Paul Smolensky. 1990. Harmonic grammar — a formal multi-level connectionist theory of linguistic well-formedness: Theoretical foundations. In *Proceedings of the Twelfth Annual Meeting of the Cognitive Science Society*, pages 388–395.

May, Robert. 1985. *Logical Form: Its Structure and Derivation*, volume 12 of *Linguistic Inquiry Monograph*. MIT Press, Cambridge, Massachusetts.

Rumelhart, D. E., G. E. Hinton, and J. L. McClelland. 1986. A general framework for parallel distributed processing. In David E. Rumelhart, James L. McClelland, and the PDP Research Group, editors, *Parallel Distributed Processing*. MIT Press, Cambridge, Massachusetts, chapter 2, pages 45–76.

Shieber, Stuart and Yves Schabes. 1990. Synchronous tree adjoining grammars. In *Proceedings of the 13th International Conference on Computational Linguistics*, Helsinki, August.

Shieber, Stuart and Yves Schabes. 1991. Generation and synchronous tree adjoining grammars. *Computational Intelligence*, 4(7):220–228.

Smolensky, P. 1986. Information processing in dynamical systems: Foundations of harmony theory. In David E. Rumelhart, James L. McClelland, and the PDP Research Group, editors, *Parallel Distributed Processing*. MIT Press, Cambridge, Massachusetts, chapter 6, pages 194–281.

Stabler, Jr., Edward P. 1991. Avoid the pedestrian's paradox. In Robert C. Berwick, Steven P. Abney, and Carol Tenny, editors, *Principle-Based Parsing: Computation and Psycholinguistics*. Kluwer Academic Publishers, Dordrecht, Holland, pages 199–237.

Steedman, Mark. 1989. Grammar, interpretation, and processing from the lexicon. In William Marslen-Wilson, editor, *Lexical Representation and Process*. MIT Press, Cambridge, Massachusetts, chapter 16, pages 463–504.

Steedman, Mark. 1992. Grammars and processors. Technical Report MS-CIS-92-52, Department of Computer and Information Science, University of Pennsylvania. To appear in H. Kamp and C. Rohrer, eds., *Current Theories in Natural Language Understanding*, Reidel, Dordrecht, Holland.

Tyler, Lorraine K. and William D. Marslen-Wilson. 1977. The on-line effects of semantic context on syntactic processing. *Journal of Verbal Learning and Behavior*, 16(6):683–692.