Explorations in ACT-R Based Cognitive Modeling – Chunks, Inheritance, Production Matching and Memory in Language Analysis

Jerry T. Ball

Air Force Research Laboratory
Wright-Patterson Air Force Base
jerry.ball.@wpafb.af.mil

Abstract

This paper explores the benefits and challenges of using the ACT-R cognitive architecture in the development of a large-scale, functional, cognitively motivated language analysis model. The paper focuses on chunks, inheritance, production matching and memory, proposing extensions to ACT-R to support multiple inheritance and suggesting a mapping from the focus of attention, working memory and long-term memory to ACT-R buffers and declarative memory (DM).

Introduction

Our research team has been working on the development of a language analysis model (Ball, 2011; Ball, Heiberg & Silber, 2007) within the ACT-R cognitive architecture (Anderson, 2007) since 2002 (Ball, 2004). The focus is on development of a general-purpose, large-scale, functional model (Ball, 2008; Ball et al., 2010) that adheres to well established cognitive constraints on human language processing (HLP) as realized by ACT-R.

This paper explores the benefits and challenges of using the ACT-R cognitive architecture in the development of the language analysis model. ACT-R is a hybrid symbolic, subsymbolic (or probabilistic) architecture which combines parallel, probabilistic mechanisms for declarative memory (DM) chunk activation and selection (i.e. retrieval), and a parallel, utility based production matching and selection mechanism, with a serial production execution mechanism. The production system is the central component of ACT-R. It interfaces to DM and other cognitive/perceptual modules (e.g. motor module, visual module) via a collection of module specific buffers which contain chunks that constitute the current context for production matching. Buffers are restricted to containing a single chunk at a time. The production with the highest utility which matches the current context is selected and executed. Execution of a production may effect an action resulting in a change to the current context (e.g. via retrieval of a DM chunk, or a shift in attention to a new visual object). The changed context determines which production next matches and executes.

The paper focuses on declarative memory (DM) chunks and chunk types, inheritance, production matching and selection, and the mapping of ACT-R architectural features like buffers and DM to memory constructs like the focus of attention, working memory and long-term memory.

To support the creation of integrated linguistic representations, the language analysis model uses a collection of buffers—an extension of the ACT-R architecture—to retain the partial products of language analysis. These buffers are functionally needed to support language analysis. We view the collection of buffers, in combination with productions which match against them, as the ACT-R/language analysis model equivalent of Baddeley's Episodic Buffer (Baddeley, 2000). According to Baddeley, "The episodic buffer is assumed to be a limited-capacity temporary storage system that is capable of integrating information from a variety of sources…the buffer provides not only a mechanism for modeling the environment, but also for creating new cognitive representations" (ibid, p. 421). A key empirical result which motivated Baddeley to introduce the episodic buffer after 25 years of working memory research is the *prose recall* effect. Subjects are capable of recalling greater than 16 words (without error) in the context of a text passage, whereas they are limited to recalling about 5 words (without error) in isolation. This prose recall capability greatly exceeds the capacity of the phonological loop and appears to be based on a chunking (or integration) mechanism. Evidence of a prose recall effect in patients with a severely degraded long-term memory (LTM) capacity argues against an LTM explanation (LTM ~ DM in the ACT-R architecture). Ultimately, the empirical evidence led Baddeley to propose the addition of the episodic buffer to his model of working memory.

Currently, the language analysis model comprises just under 60,000 declarative memory (DM) chunks (including 59,000 lexical item chunks) and ~750 (grammatical) productions. The model is capable of processing a broad range of English language constructions (Ball, Heiberg & Silber, 2007; http://doublertheory.com/comp-grammer/comp-grammar.htm). Our ultimate goal is to be able to handle unrestricted text at levels comparable to leading computational linguistic systems (cf. Collins, 2003) without extensive training on any specific corpus—as is true

of humans, but not computational linguistic systems which require extensive training on specific, annotated corpora. In terms of processing speed, the model is capable of processing ~151 written words per minute (wpm) on a quad-core, 64-bit machine running Windows 7 and Allegro Common Lisp (ACL) (64-bit software versions). In addition, ACT-R supports the measurement of cognitive processing time, and the model is capable of processing ~150 wpm in ACT-R cognitive processing time. This compares to a cognitive processing range of between 200 and 300 wpm for adult readers of English (for full comprehension). To bring the model into closer alignment with adult reading rates, we are working on reducing both the number of productions which must execute, and the number of DM elements which must be retrieved, during language analysis (Freiman & Ball, 2010).

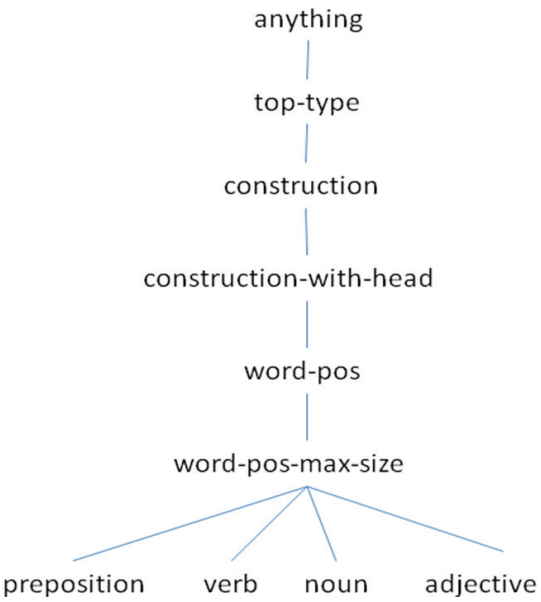
Chunks, Inheritance & Production Selection

ACT-R provides support for a hierarchy of chunk types with a single inheritance mechanism. Default values can be set for the values of slots. These default values may be overridden in specific chunks. For example, in our model, a verb chunk type is defined that inherits from a word-pos chunk type (i.e. word form + part of speech). The verb chunk type inherits word form slots from word-pos (e.g. letter and trigram slots) along with default values (all are set to "none" by default). The verb chunk type specifies a collection of slots that are common to all verbs which include "tense", "aspect", "voice", and "verb-form". The default values for these slots are also "none". A specific verb chunk overrides these default values to provide verb specific values. A sample chunk (present tense, base form of the verb "get") is shown below:

```
get-verb-pres-base-wf-pos
isa verb
index get-indx
word get-word
word-form "get"
word-symbol get-symb
word-length three3
letter-1 g
letter-2 e
letter -3 t
trigram-1 wbge ;; wb = word boundary
trigram-2 get
trigram-3 etwb
parent "none"
token "type"
stem "get"
super-type verb
type verb
subtype trans-verb
alt-subtype got-passive
subj-agree plur
verb-form v-base-fin
tense pres tense-1 fin
aspect "none"
voice act
gram-func-1 pred-head
gram-func-2 "none"
subj-agree-alt first-sing
```

"Get-verb-pres-base-wf-pos" is the chunk name. "Isa verb" defines the chunk type. These entries are followed by a collection of slot name, slot value pairs. (There are actually 12 letter and 6 trigram slots, but only 3 have values other than "none".) For example, "index" is a slot name and "get-indx" is the value of the slot. If the value of a slot is a symbol (i.e. the value is not a number or a quoted string), it corresponds to the name of an ACT-R chunk. "Get-indx" (which is not quoted above) corresponds to the name of an ACT-R chunk. It is a feature of ACT-R that only slot values which are chunks can spread activation (i.e. numbers and strings do not spread activation).

We make extensive use of ACT-R's single inheritance mechanism, both in the definition of our grammatical ontology and in the production matching and selection process. For example, the full hierarchy for defining parts of speech is shown below (note that word-pos is only one subtype of construction-with-head, not all parts of speech are shown, and it is actually word-pos-max-size that specifies the 12 letter and 6 trigram slots, not word-pos):



In ACT-R, the production selection and execution process is based on the matching of the left hand side of productions against the chunks in buffers. These buffers provide the context for production selection. During the production selection process, ACT-R determines which productions currently match the buffer contents. The production with the highest utility which matches the current buffer contents is selected for execution. Productions may selectively match against any number of buffers, from 0 (in which case the production may always match) to all the buffers. Productions which match against a chunk in a buffer must

specify the type of the chunk being matched. The match to the type succeeds if the chunk in the buffer is of the specified type (e.g. *isa verb* in the production matches *isa verb* in the buffer) or the specified type is a super-type of the chunk in the buffer (e.g. *isa word-pos* in the production matches *isa verb* in the buffer). We use the capability to match to a type or super-type extensively. Specialized productions match to chunks in buffers of a very specific type (e.g. *isa verb* in the production), whereas a more general production may match the same chunk as a high level super-type (e.g. *isa construction* in the production). Specialized productions have higher utility than general productions since they are more likely to be useful in a matching context than the more general production. Productions which match chunks in buffers may also match the slot names and slot values of the chunks in the buffer—where slot values may be chunk names, numbers or strings.

The value of a slot may be the name of a chunk. The name functions like a pointer to the chunk. For example, the value "get-indx" for the slot "index" could be used to retrieve the DM chunk corresponding to the chunk name "get-indx". Using the chunk name to retrieve a chunk is a highly constrained from of retrieval—there is only one chunk that can match the chunk name. It is important to note that the contents of a chunk are not directly available from the chunk name. Access to the contents of the chunk requires a chunk retrieval. In this respect, ACT-R has a highly constrained capability for accessing the contents of a chunk and its chunk descendents. The contents of the chunks which are the slot values of a chunk are not directly accessible without a retrieval. We refer to this constraint as *local accessibility*.

The language analysis model uses ACT-R productions to support language analysis. The production matching behavior of ACT-R can be compared to other approaches to language analysis. Within computational linguistics, there is an approach to language analysis which is based on the unification of attribute-value matrices. This approach is a key element of Head-Driven Phrase Structure Grammar (HPSG) (Sag, Wasow & Bender, 2003) and certain variants of Construction Grammar (e.g. Sign-Based Construction Grammar, Sag 2010). A key difference between such approaches and our ACT-R based language analysis model is that the value of an attribute (where attribute corresponds to a slot in ACT-R terms) is itself an attribute-value matrix. Instead of the attribute having a value which is the name of an attribute-value matrix, the matrix itself is the value of the attribute. By comparison, ACT-R introduces a level of indirection in that the value of a slot is the name of a chunk and not the chunk itself. Combining the direct integration of attribute-value matrices as the value of an attribute with the unification algorithm for matching attribute-value matrices provides a powerful language analysis capability. Unlike ACT-R's limited local accessibility pattern matching capability, unification provides the capability to recursively unify arbitrarily complex attribute-value matrices. While this might appear to make approaches like HPSG preferable to our approach, I believe that this unification capability is too powerful to match human capabilities. In a similar vein, Vosse & Kempen (2000) limit the unification capabilities of their psychologically plausible language analysis system to a single level of recursion, comparable to the local accessibility of ACT-R. Further, there is an approach to formal semantic analysis called Minimal Recursion Semantics (Copestake et al., 2005) which has been used in combination with both HPSG and SBCG. Minimal Recursion Semantics limits the recursive capabilities of the semantic analysis component to a single level, using pointer variables within semantic representations to link to non-atomic elements of the semantic representation (and restricting the representations to be compliant with first order predicate logic in the process). In using pointer variables, the representations in Minimal Recursion Semantics are closer in form to the representations of our ACT-R based language analysis model than HPSG or SBCG. In addition, Sag (2007) introduces an explicit locality constraint into SBCG which limits grammatical schemas to accessing the content of the daughters of a phrase level representation, precluding access to lower levels of representation (i.e. daughters of the daughter). The introduction of locality constraints is a principle motivator of Sag's shift from HPSG to SBCG: "This paper proposes a modification of HPSG theory – Sign-Based Construction Grammar – that incorporates a strong theory of both selectional and constructional locality" (Sag, 2007). Information that is needed at a particular level of representation must be passed up from lower levels to be locally accessible. The language analysis model adopts a similar approach. For example, grammatical features like tense and aspect are lexically realized by verbs. For these grammatical features to be accessible at the level of a clause, they must be passed up or *projected* from the level of the verb to the level of the clause. Whereas it is possible in HPSG, and possible but discouraged in SBCG to use unification to recursively descend to lower levels to match grammatical features without projecting them up, the constraints of ACT-R preclude this without explicitly retrieving the lower level chunks in order to access features that have not been projected.

As noted above, the pattern matching capabilities of ACT-R's production selection mechanism are limited to matching slot values (of chunks in buffers) which may contain the name of a chunk, strings or numbers, whereas the matching to chunk types uses ACT-R's inheritance mechanism. There are cases where it is desirable to be able to match slot values in a hierarchical manner. For example, the language analysis model has an animacy feature of nouns and object referring expressions (or nominals) that has the possible values *inanimate*, *animate* and *human* (where *human* is a subtype of *animate*). For generality, we would like a production to be able to match either the *human* or *animate* feature, when appropriate. For example, when incrementally identifying the indirect or direct object of a verb like "give" (e.g. "he gave the man…" vs. "he gave

the book…"), animacy is key, with the indirect object typically being *human* or *animate*, and the direct object typically being *inanimate*. We would like a single production to handle the indirect object case and a single production to handle the direct object case. If a test for animacy matched either *animate* or *human* (a subtype of *animate*), only a single production would be needed for the indirect object case. ACT-R does not support this. A less desirable workaround is to include a disjunctive match for either *animate* or *human*. ACT-R also does not support disjunctive matching. However, ACT-R does support conjunctive negative matches. For example, the direct object production could have separate tests for the animacy feature not being either *animate* or *human* (e.g. - *animacy animate*, - *animacy human*, instead of *animacy inanimate*). In general, we avoid negative tests (and especially negative conjunctive tests) on plausibility grounds—resorting to such tests only to avoid a proliferation of productions with independent positive tests. More typically, we step outside the ACT-R architecture and add a function to the production (using ACT-R's !eval! mechanism) which performs a disjunctive test using Lisp code.

Multiple Inheritance

For reasons that have not been make explicit, but are not likely to be theoretical in any case, ACT-R only supports single inheritance. The only reason I have been able to come up with, is, if you want multiple inheritance in ACT-R (or single inheritance for that matter), you should implement it in the production system itself. In fact, there is an ACT-R tutorial unit which implements a semantic net and uses productions to provide an inheritance-like capability. However, implementing an inheritance capability within the production system would complicate model development, lead to an undesirable proliferation in the number of productions, and increase cognitive processing time.

As an example of production proliferation, consider how production matching could be implemented without using ACT-R's built-in single inheritance capability. Instead of a production matching on a super-type which encompasses a collection of types, a separate production could be created for each type. In the case of part of speech chunks, instead of a generic production that matches all *word-pos* chunks, one production could match *verbs*, another production could match *nouns*, a third production could match *adjectives*, etc. This collection of part of speech specific productions is the computational equivalent of a single production that matches the *word-pos* super-type. Note that there is no cognitive cost associated with this type matching capability since production selection occurs in parallel in ACT-R and incurs no time cost. At least for production matching and selection, using ACT-R's single inheritance mechanism to match a super-type can be viewed as a computational short cut that reduces the number of productions. To implement production matching with multiple inheritance, additional productions are still needed.

Besides having multiple productions for each type of a super-type, productions could be created to provide support for inferring a super-type or subtype from a type. ACT-R's sample semantic memory model (unit 2 in the ACT-R 6 tutorial) provides an example of a chain production for inferring the super-type of a type. However, each execution of this chain production requires 50 msec to execute, and ascending (or descending) our chunk hierarchy would increase the cognitive processing time of the model which is already slower than adult human reading rates. In addition, the empirical evidence for the time cost of inferring the super-types of a type is moot. Human memory does not appear to be organized into a neat hierarchy of super-types, types and subtypes (cf. Collins & Loftus, 1975). As an aside, the ACT-R semantic memory model is one of a few tutorial models which is not compared to human data.

There are many examples of categories that are best viewed as inheriting from multiple super-types. A classic example is that of the "pet cat". A "pet cat" is at once a pet and a cat. In a single inheritance system, a "pet cat" must be categorized as either a subtype of pet or a subtype of cat. If pet cats are categorized as cats, then for each pet cat, there will need to be a slot in the cat type to indicate that it is a "pet cat". This approach misses the generalization that being a pet is a characteristic of all pets that could be inherited from a pet super-type, eliminating the need for a pet slot in the cat type.

A good article with motivates the need for multiple inheritance in language analysis (along with default inheritance) is Daelemans, De Smedt & Gazdar (1992). In this article, the authors argue that the representation of verbs is best handled via multiple inheritance. For example, verbs can be subcategorized on the basis of their argument structure as intransitive (no object), transitive (1 object) and ditransitive (2 objects). However, they can also be subcategorized on the basis of tense and aspect as past tense (e.g. "went"), present tense (e.g. "go", "goes"), present participle (e.g. "going") and past participle (e.g. "gone"). A multiple inheritance hierarchy can capture these different categorizations efficiently. Without multiple inheritance, if argument structure is used to subcategorize verbs, then tense and aspect (i.e. the present participle expresses progressive aspect, and the past participle expresses perfect aspect) will have to be redundantly encoded for each type of verb, missing a generalization that applies to all verbs.

Our model diverges from the analysis of Daelemans, De Smedt & Gazdar in that we do not treat either argument structure or tense and aspect as a basis for creating subtypes of verbs. From a representational perspective, subcategorizing on the basis of argument structure is incompatible with the basic division of parts of speech into categories like noun, verb, adjective and preposition, which we view as abstract semantic categories in agreement with Cognitive Grammar (Langacker, 1987/1991). Argument structure is a different dimension of meaning for categorizing verbs than that used to categorize nouns and verbs (i.e. object words vs. action words). A common

mistake in the creation of ontologies is to change the dimension of meaning which provides the basis for creating subtypes at different levels of the ontology. For example, categorizing my pet dog "Fido" as a subtype of dog, confuses types and instances. "Fido" is an instance of a dog—a particular dog—not a subtype or subclass of dog. The Cyc Knowledge Base (Matuszek et al., 2006) makes a clear ontological distinction between instances (#$isa) and collections or classes (#$genls). This confusion is less obvious in the treatment of *intransitive verb* as a subtype of *verb*. However, consider the possibility of creating subtypes of *verb* that correspond to *action*, *event*, *process* and *state*. This seems like a more reasonable subcategorization to me than a subcategorization based on the number of arguments. Of course, multiple inheritance would make it possible to have it both ways and would reduce the strength of this argument. Wherever multiple dimensions of meaning are involved in categorization (almost always the case), support for multiple inheritance is needed.

As another example of where multiple inheritance would be useful, consider wh-words like "who" and "what". These words are pronouns, but they are also wh-words. On the other hand, "where" and "why" are typically considered to be adverbs, not pronouns. Without multiple inheritance, we have defined the chunk types *wh-pronoun* and *wh-adverb*. These chunk types inherit from *wh-word*, but not from *pronoun* or *adverb*. This means that they cannot inherit the features of pronouns and adverbs and they cannot be recognized as pronouns or adverbs when appropriate. The result is a loss of generalization which requires extra productions. There are productions which match against *pronouns*, and separate productions which match against *wh-pronouns* (which are *wh-words*, but not *pronouns*). There are also productions which match against *adverbs*, and separate productions which match against *wh-adverbs* (which are *wh-words*, but not *adverbs*). With multiple inheritance, fewer productions would be needed.

Mapping ACT-R Buffers and DM into Memory Constructs

ACT-R comes with a small collection of buffers that provide the interface between various modules (e.g. visual, manual, retrieval, imaginal, goal) and the production system. These buffers are limited to holding a single chunk—a very strong constraint! Recent research has provided neuro-scientific support (primarily fMRI based) for the existence of these buffers, demonstrating that specific regions of the brain are active when these buffers are being matched against productions. With the introduction of ACT-R 6, a capability to create new modules and buffers was added to ACT-R. There are now numerous ACT-R models that posit the existence of one or two new buffers, often providing neuro-scientific support for the buffers. Our language analysis model is unique in positing the existence of dozens of new buffers (where each buffer is limited by ACT-R to holding a single chunk). The existence of these buffers is motivated on functional grounds. I do not believe it is feasible to analyze language without retaining the partial products of that analysis in a directly accessible way that avoids the necessity of retrievals from DM. As the capabilities of our model have increased, we have needed to add buffers to retain more and more different types of information. We have limited the types of chunks that are retained in each buffer. The alternative of using a smaller set of general purpose buffers would create serious problems for production matching. With type specific buffers, productions know which buffers to match against (# of productions = # of types). With type general buffers, multiple productions would be needed to match against each type general buffer (# of productions = # of buffers times # of types). Without types (or super-types), the number of productions is even larger (# of productions = # of buffers times # of subtypes).

For an example of the need to retain the partial products of language analysis in directly accessible type specific buffers, consider the input

- What did he eat?

The wh-word "what" occurs at the beginning of the sentence, but it is also the understood object of "eat" (i.e. it is the object that was eaten that is being questioned by "what"). In order to bind "what" to the object of "eat", "what" needs to be accessible at the time "eat" is processed. The simplest solution within the constraints of ACT-R is to have a buffer that contains the chunk for "what". We call this buffer the *wh-focus* buffer. Do we think this buffer is innate? No. There are languages like Chinese which have in situ wh-words (i.e. the equivalent of "he ate what"). For such a language, a *wh-focus* buffer is not needed. This suggests that English speakers learn how to buffer wh-words at the beginning of sentences, because they need to. Note that the distance between the wh-word and the binding site can be arbitrarily far:

- What do you think he ate?
- What do you think he wants to eat?

What is not allowed is the occurrence of an intervening wh-word:

- *What do you think who wants to eat?

This can be explained if there is only room for one wh-word in the wh-focus buffer. Note also that there can be more than one wh-word in a sentence:

- Who wants what?

So long as the wh-word in the wh-focus buffer doesn't have to "hop over" another wh-word for binding purposes, it's OK. (Of course there may be languages which allow wh-words to be stacked in which case the language learner would need to learn how to retain multiple wh-words in buffers.)

One might ask why not just retrieve the wh-word from memory when needed. Unless there's some indication that a

wh-word has occurred, then there's no reason the model would attempt to retrieve a wh-word. Having the wh-word in a buffer provides this indication, as well as avoiding the need for a retrieval.

The occurrence of a wh-word at the beginning of a sentence which corresponds to an object of a subsequent verb is an example of a *long-distance dependency*. The representation and processing of long-distance dependencies is an important research topic in linguistics and computational linguistics. In a common variant of generative grammar, wh-words are moved from the object position to the fronted position and leave behind an indexed trace in the object position. In Generalized Phrase Structure Grammar (GPSG), a slash notation is used to indicate the fronting of a wh-phrase (e.g. S\NP indicates the fronting of an NP at the level of the sentence, where wh-phrases are treated as NPs with a +wh feature). This slash is propagated down the syntax tree (thru the VP\NP node) until it is eliminated at the location of the missing verb object. In our language analysis model, we use the *wh-focus* buffer to retain the wh-word until it can be bound to the *object* slot in the *transitive verb* construction projected by the verb "eat".

The originators of the ACT-R cognitive architecture do not provide a definitive mapping from commonly used memory constructs like *focus of attention*, *working memory* and *long-term memory* to basic architectural features like *buffers* and *declarative memory*. I have not come across any suggested mapping of the *focus of attention* into ACT-R architectural features. However, memory constructs have been addressed to some extent. Anderson (1980) explicitly rejects the notion of *short-term memory* as elaborated in the 1960's (Atkinson & Shiffrin, 1968), and there is some recognition of the need for an *episodic memory* capacity within the ACT-R community.

In the absence of a definitive mapping, various proposals have been made. In the case of *working memory*, it has been suggested that the contents of the *buffers* constitute *working memory* (Glenn Gunzelmann, p.c.). An alternative proposal is that all chunks in DM whose activation exceeds the retrieval threshold constitute *working memory* (Christopher Myers, p.c.). Unfortunately, the first suggestion seems too constrained and the second proposal too unconstrained as a definition of working memory. The first suggestion would limit working memory to one chunk of a given type, since buffers are module, and presumably chunk type, specific, and are limited to a single chunk. A capability to compare two chunks of a given type seems necessary for an adequate definition of working memory. (I should note that it is common practice in the ACT-R community to compare chunks across buffers, often enlisting the visual or imaginal buffer for comparison with a chunk in the retrieval buffer. I consider this practice suspect under the assumption that modules and their associated buffers and chunks are specialized for the module. Can a chunk in the visual buffer really be compared to a chunk in the retrieval buffer?) For the second proposal, unless there is an associated assumption that only chunks that are receiving some amount of spreading activation from the context will exceed the retrieval threshold, then it is not clear what *working memory* means in this case (i.e. all chunks with high base level activation will exceed the threshold despite the context).

Working memory may not, itself, be a unified construct. Ericsson & Kintsch (1995) introduce a distinction between *short-term working memory* (STWM) and *long-term working memory* (LTWM) which has not been adopted by the ACT-R community, but which I find attractive. LTWM is a construct which supports expertise, making potentially large amounts of contextual relevant knowledge highly accessible. How can this working memory distinction be folded into a mapping to ACT-R, along with a mapping to the focus on attention?

Here are my suggestions. The *focus of attention* is extremely limited in capacity—estimates range between 1 and 4 chunks (Cowan, 2005). I suggest that the *focus of attention* corresponds to the chunks in the buffers that have been matched by the currently executing production. Although there is no architectural limit on the number of buffers which can be matched by a production, I suspect that matching more than 4 buffers in a production is unlikely. Ericsson and Kintsch limit STWM to 1 or 2 chunks which suggests that STWM corresponds roughly to the *focus of attention*. In a variant of ACT-R which includes carryover activation and a resonance capability (see Ball, in preparation), LTWM may correspond to the contents of all buffers not in the focus of attention, plus the chunks in DM that are resonating. There's one complication—according to Ericsson and Kintsch, it takes time to access the contents of LTWM, but access to LTWM is much faster than access to *long term memory* (LTM) more generally (where DM ~ LTM less *episodic memory* ~ *semantic memory*). Access to ACT-R buffers is instantaneous, so there is a disconnect here if LTWM includes the contents of buffers not in the focus of attention as well as resonating chunks in DM, unless LTWM access time is an average across instantaneous buffers and slower DM. It is also the case that Ericsson and Kintsch believe that LTWM is adaptive. Experts learn how to activate larger and larger quantities of contextually relevant knowledge in LTWM where it is readily accessible. I think this ability corresponds to an ability to learn how to buffer useful knowledge in ACT-R via the creation of new buffers—although this capability to create new buffers doesn't currently exist in ACT-R. I believe this capability is chunk type specific. A learned buffer retains chunks of a particular type. I don't think Ericsson and Kintsch consider this in their description of LTWM, although it is clear that expert knowledge is domain, if not type, specific. In sum, allowing LTWM to map to a combination of learned buffers not in the focus of attention and resonating DM chunks appears to provide a good mapping, although neither of these are currently part of the ACT-R architecture.

In an interesting article on working memory, Baddeley (2002) revises his 25-year old theory of *working memory* by adding an *episodic buffer* in addition to the *phonological*

loop and *visualspatial sketchpad*. The *episodic buffer* is a temporary storage system and differs from Tulving's notion of *episodic memory* as a *long-term memory* construct in this respect. The need for this buffer is motivated, in part, by the relatively large number of words that can be accurately recalled in the context of a sentence (~16 words without error) compared to isolated words (~5 words without error). This same *prose recall* data was used by Ericsson & Kintsch to motivate their LTWM. Unlike ACT-R buffers which are limited to holding a single chunk, Baddeley's *episodic buffer* is a complex system capable of holding an as yet undetermined number of multi-modal memory structures—with initial estimates ranging between 5 and 10 words in the case of verbal information. The term *episodic* is used in the sense that "it is capable of binding together information from a number of different sources into chunks or episodes" (Baddeley, 2003, p. 203), and the term *buffer* is used in the sense of "providing a way of combining information from different modalities into a single multi-faceted code" (ibid.). Baddeley constrasts this *episodic buffer*, which is a distinct temporary storage system, with Ericsson & Kintsch's LTWM, which corresponds to activated LTM (with pointers from STWM to support efficient, direct retrieval). Baddeley also provides neuro-scientific evidence for *the episodic buffer*, referencing the research of Prabhakaran et al. (2000) and providing the following quote from that research: "the present fMRI results provide evidence for another buffer, namely one that allows for temporary retention of integrated information" (p. 89). Even more striking, Baddeley discusses a subject studied by Endel Tulving who had seriously impaired LTM, but was nonetheless a good bridge player—which requires retention of considerable information. Apparently, this subject had learned how to buffer knowledge of bridge prior to suffering a LTM impairment, and retained the ability to play bridge despite the impairment.

There is a more direct mapping from Baddeley's complex (multi-chunk) *episodic buffer* to the collection of single chunk buffers used in the language analysis model, than from Ericsson & Kintsch's LTWM when viewed as activated LTM. Chunks in buffers are treated as specific instances (i.e. episodic) rather than generic types (i.e. semantic) even though they are retrieved from ACT-R's DM—which corresponds more closely to *semantic memory* than *episodic memory*. It is unclear if LTWM is more semantic or episodic in nature. In addition, the contents of the *episodic buffer* are immediately accessible as are ACT-R buffers, unlike LTWM.

To the extent that there is empirical support for constructs like the *episodic buffer* and LTWM, and, more generally, to the extent that there is empirical motivation for a larger *working memory* than provided by the built-in ACT-R buffers, the collection of buffers used in the language analysis model is also supported.

In a recent presentation by John Anderson (June 2011), he introduced a new collection of buffers to support metacognition in ACT-R. Metacognition occurs during the completion of complex tasks and encompasses processes like reflection, high-level reasoning and theory of mind. Anderson provided fMRI evidence for these buffers based on the activation of several distinct brain regions during the performance of complex algebraic tasks. Anderson also noted that the performance of tasks which require a mapping from learned techniques for completing algebraic equations, to novel, but isomorphic, ways of representing algebraic tasks requires simultaneous maintenance of at least two chunks to perform the mapping from learned to novel representation. The introduction of these new buffers and the recognition of the need to maintain multiple chunks for comparison and analogy extends the capability of ACT-R for modeling complex tasks, and begins to address some of the large areas of the brain for which ACT-R currently has little to say (including much of the pre-frontal cortex).

In general, the introduction of new buffers in ACT-R is based on extensive empirical evidence on simple tasks. Across simple tasks, the same buffers are often reused—obviating the need to expand the number of buffers. However, as more complex ACT-R models are developed, the attempt to reuse the small set of existing buffers can lead to severe interference within individual buffers as chunks override each other. This interference can take the form of thrashing of DM retrievals as one retrieval overrides a previous chunk that is still needed, necessitating its re-retrieval, which overrides the current chunk that is also needed. The result is an inability to model human performance (i.e. too many retrievals are required which slows the model down to well below human performance). To perform complex tasks, more information is needed than can be maintained by the existing ACT-R buffers, given psychologically motivated chunk sizes. Either chunks must grow in size, or the number of buffers must be increased.

In an earlier version of our synthetic teammate model (Ball et al., 2010), we incorporated what we called a "superchunk" that contained a large number of slots to retain the information needed to pilot an Unmanned Aerial Vehicle (UAV). This superchunk was stored in ACT-R's *imaginal buffer* and allowed the model to function, but was representationally and cognitively problematic. We have since modified the model (Rodgers et al., 2011) to introduce a collection of buffers that contain chunks which are representationally more defensible, and I would argue cognitively more plausible. In complex systems, representations matter. Our chunks are still larger than standard psychological assumptions support (i.e. most psychological theories limit chunks to having 3 or 4 elements), but Ball (2011, part 1) provides arguments for why the chunks are as large as they are given ACT-R constraints on activation spread.

Conclusions

ACT-R's constraints on production matching and the limits of single inheritance, combined with assumptions about chunk size and local access to chunk contents have created

opportunities and challenges in the development of a large-scale functional language analysis model. The problems can be allayed to some extent by the addition of buffers to retain the partial products of language analysis and the creation of additional productions where multiple-inheritance is needed, but not supported. Empirical support for the addition of these buffers is provided via association with Baddeley's *episodic buffer* and, to some extent, with Ericsson & Kintsch's LTWM.

References

Anderson, J. (1980). *Cognitive psychology and its implications*. San Francisco: Freeman.

Anderson, J. (2007). *How Can the Human Mind Occur in the Physical Universe?* NY: Oxford University Press.

Anderson, J. (2011). Development of the ACT-R Theory and System. Presentation at the ONR Cognitive Science Program Review. Arlington, VA.

Atkinson, R. & Shiffrin, R (1968). Human memory: A proposed system and its control processor. In Spence, W. & Spence, J. (eds.), *The psychology of learning and motivation,* 89-195. NY: Academic Press.

Baddeley, A. (2000). The episodic buffer: a new component of working memory? *Trends in Cognitive Sciences, 4(11),* 417-423.

Baddeley, A. (2002). Is Working Memory Still Working? *European Psychologist, 7(2),* 85-97.

Baddeley, A. (2003). Working memory and language: an overview. *Journal of Communication Disorders, 36*, 189-208.

Ball, J. (2004). A Cognitively Plausible Model of Language Comprehension. *Proceedings of the 13th Conference on Behavior Representation in Modeling and Simulation*, pp. 305-316. ISBN: 1-930638-35-3

Ball, J. (2008). A Naturalistic, Functional Approach to Modeling Language Comprehension. *Papers from the AAAI Fall 2008 Symposium, Naturally Inspired Artificial Intelligence*. Menlo Park, CA: AAAI Press

Ball, J. (2011). A Pseudo-Deterministic Model of Human Language Processing. *Proceedings of the 2011 Cognitive Science Society Conference*.

Ball, J. (in preparation). Explorations in ACT-R Based Cognitive Modeling – Activation, Selection and Verification without Inhibition in Language Analysis.

Ball, J., Freiman, M., Rodgers, S. & Myers, C. (2010). Toward a Functional Model of Human Language Processing. *Proceedings of the 32nd Annual Meeting of the Cognitive Science Society.*

Ball, J., Heiberg, A. & Silber, R. (2007). Toward a Large-Scale Model of Language Comprehension in ACT-R 6. In R. Lewis, T. Polk & J. Laird (Eds.) *Proceedings of the 8th International Conference on Cognitive Modeling*. 173-179. NY: Psychology Press.

Collins, A. & Loftus, E. (1975). A spreading activation theory of semantic processing. *Psychological Review*, 82, 407-428.

Collins, M. (2003). Head-Driven Statistical Models for Natural Language Parsing. *Journal of the Association for Computational Linguistics, 29,* 589-637.

Copestake, Ann, Flickinger, Dan, Pollard, Carl J. and Sag, Ivan A. (2005) Minimal Recursion Semantics: an Introduction. *Research on Language and Computation 3(4)*, 281–332.

Cowan, N. (2005). *Working memory capacity*. NY: Psychology Press.

Daelemans, W., De Smedt, K. & Gazdar, G. (1992). "Inheritance in Natural Language Processing". *Computational Linguistics* Vol 19 (2), 205-218. Cambridge, MA: MIT Press.

Ericsson, K. & Kintsch, W. (1995). Long-term working memory. *Psychological Review, 201,* 211-245.

Freiman, M. & Ball, J. (2010). Improving the Reading Rate of Double-R-Language. In D. D. Salvucci & G. Gunzelmann (Eds.), *Proceedings of the 10th International Conference on Cognitive Modeling* (pp. 1-6). Philadelphia, PA: Drexel University.

Langacker, R. (1987/1991). *Foundations of Cognitive Grammar, Vols 1 &2*. Stanford, CA: Stanford University Press.

Matuszek, C., Cabral, J., Witbrock, M. & DeOliveira, J. (2006). An Introduction to the Syntax and Content of Cyc. In *Proceedings of the 2006 AAAI Spring Symposium on Formalizing and Compiling Background Knowledge and Its Applications to Knowledge Representation and Question Answering*, Stanford, CA, March 2006.

Prabhakaran, V., Narayanan, K., Zhao, Z. & Gabrielli, J. (2000). Integration of diverse information in working memory in the frontal lobe. *Nature Neuroscience*, 3, 85-90.

Rodgers, S., Myers, C., Ball, J. & Freiman, M. (2011). The Situation Model in the Synthetic Teammate Project. *Proceedings of the 20th Annual Conference on Behavior Representation in Modeling and Simulation.*

Sag, I. (2007). Remarks on Locality. *Proceedings of the HPSG07 Conference*.

Sag, I. (2010). Sign-Based Construction Grammar: An informal synopsis. In H. Boas & I. Sag (eds.). *Sign-Based Construction Grammar.* Stanford, CA: CSLI.

Sag, I., Wasow, T. & Bender, E. (2003). *Syntactic Theory: A Formal Introduction*. Stanford, CA: CSLI.

Vosse, T. & Kempen, G. (2000). Syntactic structure assembly in human parsing: a computational model based on competitive inhibition and a lexicalist grammar. *Cognition, 75*, 105-143.