A Computational Psycholinguistic Model of Natural Language Understanding

Jerry T. Ball

Air Force Research Laboratory, 6030 S. Kent Street, Mesa, AZ 85212-6061

Abstract. Double R Model (Referential and Relational Model) is a computational psycholinguistic model of natural language understanding founded on the linguistic principles of Cognitive Linguistics and implemented using the ACT-R cognitive architecture and modeling environment

1 Double R Grammar

Double R Grammar [1] is the Cognitive Linguistic theory [2,3] underlying Double R Model. In Cognitive Linguistics, all grammatical elements have a semantic basis, including parts of speech, grammatical markers, phrases and clauses. Our understanding of language is embodied and based on experience in the world [4]. Categorization is a key element of linguistic knowledge, and categories are seldom absolute—exhibiting, instead, effects of prototypicality, family resemblance [5], base level categories [6], fuzzy boundaries, and radial structure [7]. Our linguistic abilities derive from basic cognitive abilities—there is no autonomous syntactic component separate from the rest of cognition. Knowledge of language is for the most part learned and not innate. Abstract linguistic categories (e.g. noun, verb, nominal, clause) are learned on the basis of experience with multiple instances of words and expressions which are members of these categories, with the categories being abstracted and generalized from experience. Also learned are schemas which abstract away from the relationships between linguistic categories. Over the course of a lifetime, humans acquire a large stock of schemas at multiple levels of abstraction and generalization, representing knowledge of language and supporting language understanding. These schemas constitute what might be called **grammatical semantics** [8] in contrast to the **lexical semantics** of individual lexical items, although the schemas are, for the most part, associated with specific lexical items.

Two key dimensions of meaning that get grammatically encoded are referential meaning and relational meaning. Double R Grammar is focused on the representation and integration of these two dimensions of meaning within the wider scope of Cognitive Linguistics. Consider the expressions

1. The book on the table
2. The book is on the table

T. Ball J. (2004).
A Computational Psycholinguistic Model of Natural Language Understanding.
In *Proceedings of the 1st International Workshop on Natural Language Understanding and Cognitive Science*, pages 3-14
DOI: 10.5220/0002667400030014

These two expressions have essentially the same relational meaning. They both express the relation "on" existing between "a book" and "a table". However, their referential meaning is significantly different. Expression 1, as a whole, refers to an object and is called an **object referring expression**. In referring to an object, 1 uses the determiner "the" to **specify** that the object is salient in the context of use of the expression (and may have previously been referred to). Expression 1 also uses the word "book" to indicate the type of object being referred to, with "book" functioning as the **head** of the expression. Further, the phrase "on the table" refers to a location with respect to which the object can be identified and functions as a **modifier** in the expression. In referring to a location, the expression "on the table" refers to a second object "the table" and indicates the location of the first object with respect to the second object. Within the modifying expression, the relation "on" functions as the **relational head** with the object referring expression "the table" functioning as a **complement**. In expression 1, the relational meaning of "on" is subordinated to referential meaning with the modifying function of "on the table" dominating the relational meaning of "on". That is, although "on" is the relational head of the prepositional phrase "on the table", it is not the head of the overall expression and does not determine the semantic type of that expression.

Expression 2 refers to a situation and is called a **situation referring expression**. The auxiliary "is" provides a temporal specification for the situation, fulfilling a referential function similar to that of the determiner "the" in "the book" and "the table". The relational meaning of 2 is about "being on" and not just "being", with "on" functioning as the relational head of the situation referring expression. The relational head of a situation referring expression is called a **predicate**—reflecting the assertional function of the relational head. Note that "on" in 1 is not functioning as a predicate, since it is presupposed and not asserted. That is, relational heads of modifying expressions are not predicates, they are (modifying) **functions**. In expression 2, the object referring expression "the book" functions as the subject (argument) of "being on" with "the table" functioning as the object (argument). Referentially, there is also a reference to a location "on the table", which competes with the expression of the relational meaning of "on" as reflected in the difference between:

3. What is the book on?
4. Where is the book?

where 3 highlights the relation "on" in asking about the object of that relation and 4 highlights the reference to a location using "where" to do so.

The terms **specifier**, **head**, **modifier** and **complement** are borrowed from X-Bar Theory [9]. It is acknowledged that X-Bar Theory captures an important grammatical generalization, with the distinction between specifiers and modifiers representing a significant advance, but X-Bar theory is in need of semantic motivation, which, when provided, necessitates certain modifications to the theory [10]. For example, the combination of a specifier and a head results in a maximal projection which corresponds to a referring expression. However, the specifier determines the type of referring expression, not the head, with referential type corresponding most closely to the syntactic type of a maximal projection in X-Bar Theory. The head, on the other hand, determines the relational type of the expression (where "relational"

encompasses objects). Both referential type (from the specifier) and relational type (from the head) project to the expression as a whole. Consider the referring expression

5. The kick

in which the specifier "the" determines the expression to be an object referring expression, whereas, the head "kick" determines the expression to be a type of action. In 5, the specifier has the effect of objectifying the action expressed by "kick", allowing it to be referred to as though it were an object. Note that since the inherent relational meaning of "kick" is not affected (only its function), there is no need to assume that the part of speech of "kick" is a noun instead of a verb (i.e. the head does not project the "syntactic" type of the referring expression, the specifier does). And if we allow verbs (especially action verbs) to function as heads of object referring expressions (i.e. noun phrases), then one of the primary syntactic arguments against the meaning based definition for parts of speech is nullified. That the head of an object referring expression need not be a noun is further demonstrated by the following examples:

6. The **cheering up** (present participle + verb particle) of the crowd
7. She is an **Audrey Hepburn** (proper noun) in the rough
8. The **up and down** (conjoined prepositions) of the elevator
9. Your **giving money to strangers** (participial phrase) is very generous

Besides demonstrating that the head of an object referring expression need not be a noun, these examples show the importance of distinguishing the form of an expression from its function in a particular context.

2 Double R Process

Double R Process is the psycholinguistic theory of language processing underlying Double R Model. It is a highly interactive theory. Representations of referential and relational meaning are constructed directly from input texts. There is no separate syntactic analysis that feeds a semantic interpretation component. The processing mechanism is driven by the input text in a largely bottom-up, lexically driven manner. There is no top-down assumption that a privileged linguistic constituent like the sentence will occur. There is no phrase structure grammar and no top-down control mechanism. How then are representations of input text constructed? Operating on the text from left to right, schemas corresponding to lexical items are activated. For those lexical items which are relational or referential, these schemas establish expectations which both determine the possible structures and drive the processing mechanism. A short-term working memory (STWM) [11] is available for storing arguments which have yet to be integrated into a relational or referential structure, partially instantiated relational and referential structures, and completed structures. If a relational or referential entity is encountered which expects to find an argument to its left in the input text then that argument is assumed to be available in STWM. If the relational or

referential entity expects to find an argument to its right in the input text, then the entity is stored in STWM as a partially completed structure and waits for the occurrence of the appropriate argument. When that argument is encountered it is instantiated into the stored relational or referential structure. Instantiated arguments are not separately available in STWM. This keeps the number of separate linguistic units which must be maintained in STWM to a minimum.

3 ACT-R

ACT-R is a cognitive architecture and modeling environment for the development of computational cognitive models [12]. It is a psychologically validated cognitive architecture which has been used extensively in the modeling of higher-level cognitive processes (see the ACT-R web site for an extensive list of models and publications). ACT-R includes symbolic **production** and **declarative memory** systems integrated with subsymbolic **production selection** and **spreading activation** and **decay** mechanisms. Production selection involves the parallel matching of the left-hand side of all productions against a collection of **buffers** (e.g. goal buffer, retrieval buffer, visual buffer, auditory buffer) which contain the active contents of memory and perception. Production execution is a serial process—only one production is executed at a time. The parallel spreading activation and decay mechanism determines which declarative memory chunk is put into the retrieval buffer for comparison against productions. The combination of symbolic and subsymbolic mechanisms makes ACT-R a hybrid system of cognition. The **noise** parameter used by these computational mechanisms adds stochasticity to the system. ACT-R supports **single inheritance** of declarative memory chunks and limited, variable-based **pattern matching** (including a **partial-matching** capability). ACT-R incorporates **learning** mechanisms for learning both declarative and procedural knowledge. Version 5 of ACT-R adds a **perceptual-motor** component supporting the development of embodied cognitive models. With the addition of the perceptual-motor component, and the use of buffers as the interface between various cognitive modules (e.g. vision module, declarative memory), ACT-R is referred to as an "integrated theory of the mind" [13].

4 Double R Model

Double R Model is the ACT-R based computational implementation of Double R Grammar and Process (together called Double R Theory). Double R Model is currently capable of processing an interesting range of grammatical constructions including: 1) intransitive, transitive and ditransitive verbs; 2) verbs taking clausal complements; 3) predicate nominals, predicate adjectives and predicate prepositions; 4) conjunctions of numerous grammatical types; 5) modification by attributive adjectives, prepositional phrases and adverbs, etc. Double R Model accepts as input as little as a single word or as much as an entire chunk of discourse—using the perceptual component of ACT-R to read words from a text window. Unrecognized

words are simply ignored. Unrecognized grammatical forms result in partially analyzed text, not failure. The output of the model is a collection of declarative memory chunks that represent the referential and relational meaning of the input text. Although Double R Model is essentially a computational psycholinguistic model, it is intended to be used as the basis for development of large-scale, functional language understanding systems and the current coverage of the model will need to be extended significantly to support that objective.

4.1 Inheritance vs. Unification

Unification allows for the unbounded, recursive matching of two logical representations and is an extremely powerful pattern matching technique used in many language processing systems. Unfortunately, it is psychologically too powerful. For example, the following two logical expressions can be unified:

```
p(a,B,c(d,e,f(g,h(i,j),K),l))
p(X,b,c(Y,e,f(Z,T,U),l))
```

where capitalized letters are variables and lowercase letters are constants. Humans are unlikely to be capable of performing such unifications consciously or otherwise without significant effort and an external scratch pad, since STWM does not have the capacity to retain more than a few variable bindings simultaneously.

On the other hand, although extremely powerful, unification does not support the matching of types to subtypes. Thus, if we have a verb type with intransitive and transitive verb subtypes, unification cannot unify a chunk of type verb with a chunk of type intransitive verb or transitive verb. Unification's inability to match types to subtypes often results in a proliferation of rules (or conditions on rules) to handle the various combinations. For example, the verb type can be variableized and a test for the valid types can be used to constrain the variable (e.g. Verb-Type equal verb or Verb-Type equal intrans-verb or Verb-Type equal trans-verb). With inheritance, a production that checks for a verb type will also match a transitive verb and an intransitive verb type (assuming an appropriate inheritance hierarchy). Humans appear to be able to use types and subtypes in appropriate contexts with little awareness of the transitions. For example, when processing a verb, all verbs (used predicatively) expect to be preceeded by a subject, but only transitive verbs expect to be followed by an object. Thus, humans presumably have available a general production that applies to all verbs (or even all predicates) which will look for a subject preceding the verb, but only a more specialized production for transitive verbs (or transitive predicates) which will look for an object following the verb.

Inheritance supports the matching of two representations without requiring the recursive matching of their subparts so long as the types of the two representations are compatible. Types are essentially an abstraction mechanism which makes it possible to ignore the detailed internal structure of representations when comparing them. For example, once the model has identified an expression as an object referring expression, the model can match the object referring expression against productions without consideration of the internal structure of the expression. Of course, there may be productions that do consider the internal structure, but types are useful here as

well. Instead of having to fully elaborate the internal structure, types can be used to partially elaborate that structure. For example, if a production is specifically concerned with object referring expressions headed by a quantifier (e.g. “some” in “some of the books”), the production can check to see that the head is of the appropriate type, providing a (limited) unification like capability where needed. In sum, inheritance and limited pattern matching provide a psychologically plausible alternative to a full unification capability.

To take advantage of inheritance, Double R Model incorporates a type hierarchy (a tangled hierarchy or lattice, with multiple inheritance, is preferred, but ACT-R currently only supports single inheritance). Representative elements of the top levels of the current hierarchy of types (below `top-type`) are shown below:

```
       Lexical-type
         Noun  Adjective  Verb  Preposition  Adverb  Determiner
Quantifier Auxiliary
       Referential-type
         Head Specifier Modifier Complement
       Referring-expression-type
         Object-refer-expr Situation-refer-expr Predicate-refer-
expr
         Location-refer-expr Direction-refer-expr
       Relation-type
         Relation (with subtypes: Predicate Function) Argument
```

The more specialized a production is, the more specialized the types of the chunks in the goal and retrieval buffers to which the production matches will need to be. The most general productions match a goal chunk whose type is `top-type` and ignore the retrieval buffer chunk.

4.2 Default Rules

ACT-R’s inheritance mechanism can be combined with the subsymbolic production utility parameter—which influences production selection—to establish default rules. Since all types extend a base type, using the base type as the value of the goal chunk in a production will cause the production to match any goal chunk. If the production is assigned a production utility value that is lower than competing productions, it will only be selected if no other production matches. A sample default production is shown below:

```
(p process-default--retrieve-prev-chunk
   =goal> ISA top-type
   =context> ISA context
   state process
   chunk-stack =chunk-stack
   =chunk-stack> ISA chunk-stack-chunk
   this-chunk =chunk
   prev-chunk =prev-chunk
   ==>
   =context>
```

```
state retrieve-prev-chunk
chunk-stack =prev-chunk
+retrieval> =chunk)
```

where the parentheses reflect the underlying Lisp implementation, `p` identifies a production, `process-default--retrieve-prev-chunk` is the name of the production, `=goal>` identifies the goal chunk, `ISA top-type` is a chunk type, `=context>` identifies a context chunk, `state` is a chunk slot, `process` is a slot value, `==>` separates the left-hand side from the right-hand side and variables are preceded by `=` as in `=chunk`. This default production causes the previous chunk to be retrieved from declarative memory (using the `+retrieval>` form) if no other production is selected.

4.3 The Context Chunk and Chunk Stack

The current ACT-R environment provides only the goal and retrieval buffers to store the partial products of language comprehension, although earlier versions of ACT-R provided a goal stack. The lack of a stack is particularly constraining, since a stack is the primary data structure for managing the kind of (limited) recursion that occurs in language. There needs to be some mechanism for retrieving previously processed words from STWM in last-in/first-out order during processing (subject to various kinds of error that can occur in the retrieval process). A stack provides this (essentially error free) capability. It is expected that a capacity to maintain about 5 separate linguistic chunks in STWM is needed to handle most input—supporting at least one level of recursion (and perhaps two for the more gifted). The goal chunk could be adapted for this purpose, except that it is also the basis for creation of new declarative memory chunks and activation spread and these architectural needs would conflict. Further, it would be difficult to get the kind of stack like behavior needed out of the slots in the goal chunk.

To overcome these problems, Double R Model introduces a context chunk containing a bounded, circular stack of links to declarative memory. As chunks are stacked in the circular stack, if the number of chunks exceeds the limit of the stack, then new chunks replace the least recently stacked chunks (supporting at least one type of STWM error). The actual number of chunks allowed in the stack is specified by a global parameter. This parameter is settable to reflect individual differences in STWM capacity. Chunks cannot be directly used from the stack. Rather, the chunk on the stack provides a template for retrieving the chunk from declarative memory. Since the chunk must be retrieved from declarative memory before use, the spreading activation and partial matching mechanisms of ACT-R are not circumvented and retrieval errors are possible—unlike the goal stack of earlier versions of ACT-R (which was criticized for this reason). Thus, the bounded, circular stack of links to declarative memory avoids the arguments against the goal stack of earlier versions of ACT-R, adds the insight of activated pathways to declarative memory, and retains the insights that motivated the inclusion of a goal stack in the earlier versions.

Besides storing the chunk stack, the context chunk is also used to separate out state information from the goal chunk. Since the goal chunk is the basis for creating new declarative memory chunks, storing the chunk stack in it would result in the chunk stack being stored with each new declarative memory chunk. While this might be

used to support a kind of episodic memory where the context in which a word occurs is stored with the declarative memory chunk created during the processing of the word, ACT-R 5.0 does not currently provide a mechanism for transitioning episodic memory into semantic memory (i.e. abstracting from the context of use), and storing the context with a chunk has undesirable side-effects within the ACT-R environment (e.g. it interferes with the spreading activation mechanism). To avoid such problems a separate context chunk is maintained and made available to all productions. Although, the existence of a separate context chunk that productions match to violates the ACT-R 5.0 architecture where only the buffers are supposed to be used for this purpose, earlier versions of ACT-R allowed multiple chunks to be matched on the left-hand side of productions and this functionality is still available in ACT-R 5.0 environment.

The context chunk maintains several pieces of information in addition to the chunk stack. Its definition (as specified by a `chunk-type`) in the model is shown below:

```
(chunk-type context state rel-context sit-context text-context
    word prev-word-1 prev-word-2 repeat chunk chunk-stack)
```

In this `chunk-type` definition, `context` is the name of the chunk, `state` is a slot that provides state information to guide production selection, `rel-context` is a slot that identifies the current relational context, `sit-context` is a slot that contains information about the current situation context, `text-context` is a slot that contains information about the larger discourse context, `word` contains the lexical item being processed, `word-prev-1` and `word-prev-2` contains the previous two words processed, `repeat` is `yes` if the word has been attended to previously and `no-more` if there are no more words in the input, `chunk` contains the most recently processed chunk, and `chunk-stack` contains the entire chunk stack.

4.4 Lexical and Functional Entries

The lexical entries in the model provide a limited amount of information which is stored in the `word` and `word-info` chunks. The definition of the `word` and `word-info` chunk types are provided below:

```
(chunk-type word word-form word-marker)
(chunk-type word-info word-marker word-root word-type word-subtype word-morph)
```

The `word-form` slot of the `word` chunk contains the physical form of the word (represented as a string in ACT-R); the `word-marker` slot contains an abstraction of the physical form. The `word-root` slot contains the value of the root form of the word. The `word-type` slot contains the lexical type of the word and is used to convert a `word-info` chunk into a `lexical-type` chunk for subsequent processing. A `word-subtype` slot is provided as a workaround for the lack of multiple inheritance in ACT-R 5.0. The `word-morph` slot supports the encoding of morphological information

Sample lexical entries for a `noun` and `verb` are provided below:

```
   (cow-wf isa word word-form "cow" word-marker cow)
   (cow isa word-info  word-marker cow word-root cow word-type
noun
       word-morph 3d-per-sing))
   (running-wf  isa  word  word-form  "running"  word-marker
running)
   (running isa word-info word-marker running word-type verb
word-root run
       word-subtype intrans-verb word-morph pres-part)
```

Note that there is no indication of the functional roles (e.g. head, modifier, specifier, predicate, argument) that particular lexical items may fulfill. Following conversion of `word-info` chunks into `lexical-type` chunks, functional roles are dynamically assigned by the productions that are executed during the processing of a piece of text. Since functional role chunks are dynamically created, only `chunk-type` definitions exist for functional categories prior to that processing. As an example of a `chunk-type` definition for a functional category, consider the category `pred-trans-verb` (transitive verb functioning as a predicate) whose definition involves several hierarchically related `chunk-types` as shown below:

```
  (chunk-type top-type head)
  (chunk-type (rel-type (:include top-type)))
  (chunk-type  (pred-type  (:include  rel-type))  subj  spec  mod
post-mod)
  (chunk-type (pred-trans-verb (:include pred-type)) obj)
```

The `top-type chunk-type` contains the single slot `head`. All types are subtypes of `top-type` and inherit the `head` slot. `Rel-type` is a subtype of `top-type` that doesn't add any additional slots. `Pred-type` is a subtype of `rel-type` that adds the slots `subj`, `spec`, `mod`, and `post-mod`. It is when a relation is functioning as a predicate that these slots become relevant. `Pred-trans-verb` is a subtype of `pred-type` that adds the slot `obj`. Summarizing, `pred-trans-verb` contains the slots `head`, `subj`, `spec`, `mod` (pre-head), `post-mod` (post-head), and `obj`, all of which are inherited from parent types except for the `obj` slot.

The following production creates an instance of a `pred-trans-verb` providing initial values for the slots:

```
(p process-verb--pred-trans-verb
 =goal> ISA verb
   head =verb
   subtype trans-verb
   =context> ISA context
   state convert-verb-to-pred-verb
   ==>
   +goal> ISA pred-trans-verb
   subj none
   spec none
   mod none
   head =goal
```

```
    post-mod none
    obj none
    =context>
    state retrieve-prev-chunk)
```

In this production, a `verb` (subtype of `lexical-type`) whose `subtype` slot has the value `trans-verb` is converted into a `pred-trans-verb` for subsequent processing. The only slot of `pred-trans-verb` that is given a value other than `none` is the `head` slot whose value is set to be the `goal` chunk (`head =goal`). This production has the effect of assigning a transitive verb the functional role of predicate (specialized as a transitive verb predicate). Its selection and execution is based on the previous context which set the value of the `state` slot of the `context` chunk to `convert-verb-to-pred-verb` and on having a goal chunk of type `verb` whose `subtype` slot has the value `trans-verb`.

4.5 Productions

Sample productions were shown above in the discussion of default rules and in the creation of functional roles. This section discusses the productions used in the processing of the word "kick" following the word "the" in the expression "the kick". The `read-next-word` production initiates the find-attend-encode sequence for reading the next word from the computer screen (using ACT-R's perceptual component). Following the `read-next-word` production, the `retrieve-word-info` production retrieves the `word-info` chunk. The `word-info` chunk is then used by the `convert-word-to-verb` production to create a `verb lexical-type` chunk which becomes the goal. Then, the `process-verb--obj-context--convert-to-rel-head` production matches a `verb` goal chunk and in the context of an `obj` (object referring expression) converts the `verb` type into a `rel-head` type (using the `+goal>` form)

```
(p process-verb--obj-context--convert-to-rel-head
   =goal> ISA verb
   head =verb
   =context> ISA context
   state retrieve-prev-chunk
   rel-context obj
   ==>
   +goal> ISA rel-head
   mod none
   head =goal
   post-mod none)
```

Rel-head (relational head) is a subtype of head. The process-head--prev-chunk-is-obj-spec production matches a head goal chunk (which could be a rel-head) and an obj-spec (object specifier) retrieval chunk and creates a new obj-refer-expr (object referring expression) which becomes the goal. Together, these two productions support to use of verbs as (relational) heads of object referring expressions following an object specifier.

```
(p process-head--prev-chunk-is-obj-spec
   =goal> ISA head
   =context> ISA context
   state retrieve-prev-chunk
   =retrieval> ISA obj-spec
   ==>
   +goal> ISA obj-refer-expr
   spec =retrieval
   mod none
   head =goal
   post-mod none
   referent none-for-now
   =context> state process
   rel-context none)
```

The creation of an object referring expression causes the value of the `rel-context` (relational context) slot to be set to `none` indicating the end of the object referring expression context.

4.6 Context Accommodation vs. Backtracking

Context accommodation is a mechanism for changing the function of an expression based on the context without backtracking. For example, when an auxiliary verb like "did" occurs it is likely functioning as a predicate specifier as in "he did not run" where the predicate is "run" and "did not" provides the specification for that predicate. However, auxiliary verbs may also function as predicates when they are followed by an object referring expression as in "he did it". Determining the ultimate function of an auxiliary can only be made when the expression following the auxiliary is processed. In a backtracking system, if the auxiliary is initially determined to be functioning as a predicate specifier, then when the noun phrase "it" occurs, the system will backtrack and reanalyze the auxiliary, perhaps selecting the predicate function on backtracking. However, note that backtracking mechanisms typically lose the context that forced the backtracking. Thus, on backtracking to the auxiliary, the system has no knowledge of the subsequent occurrence of a noun phrase to indicate the use of the auxiliary as a predicate. Thus, the system can only randomly select a new function for the auxiliary which may or may not be that of a predicate.

A better alternative is to accommodate the function of the auxiliary in the context which forces that accommodation. In this approach, when the noun phrase "it" is processed and the auxiliary functioning as a predicate specifier is retrieved, the function of the auxiliary can be accommodated in the context of a subsequent noun phrase to be a predicate. Context accommodation avoids the need to backtrack and allows the context to adjust the function of an expression just where that accommodation is supported by the context. Of course, there may cases where the context accommodation mechanism breaks down and some form of backtracking is needed (e.g. garden-path sentences), but in such cases backtracking is likely to involve a jump back to the beginning of a major constituent (e.g. clause) and some contextual information will be carried back with the jump. In any case, a reverse-

depth-first, context-unraveling backtracking mechanism like that provided in Prolog is psychologically implausible.

Context accommodation assumes the activation, selection and integration of the most appropriate schema given the current context, subject to accommodation based on the subsequent context. Context accommodation is highly compatible with **Preference Semantics** [14], and naturally handles the cases where the initially preferred choice turns out not to be appropriate in the wider context.

5 Summary

Double R Model may be the first attempt at the development of an NLU system founded on the principles of Cognitive Linguistics and implemented using the ACT-R cognitive architecture and modeling environment. In its current state it demonstrates the possibility of building such a system. Much work remains to be done before the feasibility of building functional NLU systems using this approach can be fully demonstrated. For more details of the theory and the full source code, see the Double R Theory web site at www.DoubleRTheory.com.

References

1. Ball, J. (2003). "Double R Grammar." http://www.DoubleRTheory.com/DoubleRGrammar.pdf
2. Langacker, R. (1987, 1991). *Foundations of Cognitive Grammar, Vols 1 and 2*. Stanford, CA: Stanford University Press.
3. Talmy, L. (2003). *Toward a Cognitive Semantics, Vols I and II.* Cambridge, MA: The MIT Press
4. Lakoff, G., & M. Johnson (1980). *Metaphors We Live By.* Chicago: The University of Chicago Press.
5. Wittgenstein, L. (1953). *Philosophical Investigations*. New York: MacMillan.
6. Rosch, E. (1978). "Principles of Categorization." In *Cognition and Categorization*. Edited by E. Rosch & B. Lloyd. Hillsdale, NJ: LEA.
7. Lakoff, G. (1987). *Women, Fire and Dangerous Things.* Chicago: The University of Chicago Press.
8. Nirenburg, S. & Raskin, V. (1995). "Ten Choices for Lexical Semantics". Memoranda in Computer and Cognitive Science, MCCS-96-304. CRL, NMSU, Las Cruces, NM.
9. Chomsky, N. (1970). "Remarks on Nominalization." In R. Jacobs & P. Rosembaum, eds., *Readings in English Transformational Grammar*. Waltham, MA: Ginn.
10. Ball, J (2003). "Towards a Semantics of X-Bar Theory." http://www.DoubleRTheory.com/SemanticsOfXBarTheory.pdf
11. Kintsch, W. (1998). *Comprehension, a Paradigm for Cognition.* New York, NY: Cambridge Univ Press.
12. Anderson, J. & Lebiere, C. (1998). *The Atomic Components of Thought.* Mahway, MJ: LEA.
13. Anderson, J., Bothell, D., Byrne, M., LeBiere, C (in press). *"*An Integrated Theory of the Mind" *Psychological Review*
14. Wilks, Y. (1975). "A preferential, pattern-seeking semantics for natural language inference." *Artificial Intelligence*, 6:53-74