Piece of Mind: Long-Term Memory Structure in ACT-R and CHREST

Martyn Lloyd-Kelly (martynlk@liverpool.ac.uk)
Fernand Gobet (fernand.gobet@liverpool.ac.uk)
Department of Psychological Sciences, University of Liverpool,
Bedford Street South, Liverpool, L69 3BX, UK

Peter C. R. Lane (peter.lane@bcs.org.uk)
School of Computer Science, University of Hertfordshire,
College Lane, Hatfield, AL10 9AB, UK

Abstract

Creating a plausible Unified Theory of Cognition (UTC) requires considerable effort from large, potentially distributed, teams. Computational Cognitive Architectures (CCAs) provide researchers with a concrete medium for connecting different cognitive theories to facilitate development of a robust, unambiguous UTC. However, due to wide dissemination of research effort, and broad scope of cognition as a psychological science, keeping track of CCA contributions is difficult.

We compare the structuring of long-term memory (LTM) in two CCAs: ACT-R and CHREST. LTM structuring is considered in particular since it is an essential component of CCAs and underpins most of their operations. We aim to consolidate knowledge regarding LTM structuring for these CCA's and identify similarities and differences between their approaches. We find that, whilst the architectures are similar in a number of ways, providing consensus for some concepts to be included in a UTC, their differences highlight important questions and development opportunities.

Keywords: ACT-R, CHREST, Cognitive Architectures, Long-term Memory

Introduction

Several CCAs are currently available to psychologists; some benefit from large user-bases and development teams aiming to create pan-optic models of cognition, of which ACT-R (Anderson, 2007) is a notable example. Others, however, have a relatively smaller community and focus upon particular aspects of cognition: CHREST (Gobet & Lane, 2010) focuses on modelling general mechanisms that govern the interplay between perception and cognition resulting in learning and acquisition of expertise in disparate domains such as games, physics, language and concept formation[1]. CCAs are powerful tools that force psychologists to specify theories of cognition unambiguously, facilitating the testing and refinement of general cognitive principles in domains that apply large numbers of constraints (Newell, 1990). They also, crucially, allow psychologists to analyse the overlap and disparity between different theories of cognition.

In this paper, we delineate how the latest versions of ACT-R and CHREST (6.0 and 5.0, respectively) structure LTM with a focus on LTM topology[2]. The purpose of our comparison is three-fold: first, it offers psychologists investigating theories of LTM structuring a centralised location for current[3] information regarding how ACT-R and CHREST structure LTM. This will facilitate both understanding and efficient comparisons of the similarities and differences between the two architectures with respect to this feature and allow psychologists using ACT-R or CHREST to tailor their investigations accordingly. Second, the comparison highlights novel ways of developing both architectures and should foster dialogue and exchange of ideas between cognitive psychologists in general and the ACT-R and CHREST development groups in particular. Third, by identifying common and disparate elements of LTM structuring in both architectures, a consensus upon the subject can begin to be formalised to some degree, enabling the construction of a valid UTC.

ACT-R and CHREST are, respectively, examples of top-down and bottom-up approaches to cognitive modelling: ACT-R is inherently pluralistic (Jilk, Lebiere, O'Reilly, & Anderson, 2008) and can accommodate a number of cognitive theories. It therefore adopts a *laissez-faire* attitude towards how its structures and functions operate. CHREST, on the other hand, focuses on modelling learning and the development of expertise, and contains a number of hard-coded limitations on how its structures and functions operate. Consequently, to find some common ground between the two architectures, we focus on the architecture of ACT-R *as-is* (Bothell, n.d.), rather than considering it in accordance with any particular implementation.

The paper is structured as follows: sections *ACT-R* and *CHREST* discuss, in detail, the architectures' mechanisms for LTM organisation to provide a centralised location for current information regarding LTM structuring in CHREST and ACT-R. The *Architecture Comparison* section then performs a comparison based upon the content of the previous two sections allowing for the identification of similarities and differences between the architectures approach to structuring LTM. Finally, the *Conclusions and Future Work* section briefly summarises the contributions of the paper and outlines our plans for future work.

ACT-R

ACT-R is composed of a number of core *modules* that encapsulate the operations of particular regions of the human brain and is primarily a *production-rule system*, since all in-

[1] For an overview, see Gobet et al. (2001).

[2] "Topology" is defined here in the sense of a physical network topology.

[3] At the time of writing.

put/output from core modules must pass through a central production system using module-specific buffers as an interface. ACT-R has simulated cognition in a wide array of domains, including games (Martin, Gonzalez, & Lebiere, 2004), arithmetic (Lende & Taatgen, 2012) and language (Oliva, Serrano, del Castillo, & Ángel Iglesias, 2010).

LTM in ACT-R is embodied as *declarative* or *procedural* information that is handled by the declarative module or procedural system, respectively (Anderson, 2007). ACT-R uses *chunks* (Chase & Simon, 1973) as currency for these modules which include a reference and vectors of slots, one of which defines the chunk's type. Chunk references act as pointers to LTM information and facilitate memory retrieval; taken in isolation, references offer little practical information to modules. Chunk types are mutable, dictate what slots the chunk has (along with their default values) and can be super or subclasses of other chunk types. This enables construction of chunk type hierarchies that enable slot name and slot value inheritance. Values for slots may contain references to other chunks or simple information, such as a number. In version 6.0 of ACT-R, the slots for a chunk type can be extended *statically* or *non-statically*; this has implications when managing declarative memory information (discussed below).

ACT-R contains a mathematical, *sub-symbolic* system that underpins the declarative and procedural modules (Bothell, n.d.). This system governs what chunks are returned and how quickly (by using *activation levels* for declarative memory and *utilities* for productions) after external input is presented to ACT-R (Anderson, 2007). To determine what is returned, the greater the activation level for a chunk in declarative memory or the value of a production's utility, the more likely it is to be retrieved or selected after it has been found in LTM. To determine how quickly a chunk or production is selected, the activation level or utility affects the simulated time taken but does not influence real world time. Since the sub-symbolic system does not affect the topology of LTM, it is not considered in detail here.

Declarative Module

The declarative module maintains chunks used by ACT-R. Chunks can be created by any module at any time and all those that remain in module buffers at the conclusion of an ACT-R cycle are collected and added to the declarative module instantly. Chunk learning can occur at model compile-time (initial memories) or run-time but is, in either case, absolutely *concurrent* but usually *incremental* when applied in an ACT-R model. This temporal dissonance arises due to ACT-R's sub-symbolic system; a new chunk has all its information added concurrently to the declarative module but, if its assigned activation is below that of the defined retrieval threshold, only part of the chunk may be retrieved (if at all).

The structure of declarative memory is distinctly graph-like, since the only links that exist between chunks are slot value references; a slot value for a chunk, C, may reference another chunk C'. Retrieval of LTM consists of performing a non-directed search through LTM for a matching chunk and takes a simulated period of time (dictated by the parameters of the sub-symbolic system). Two simple methods exist to modify information in declarative memory: chunk addition or chunk merging, both of which are performed *instantaneously*.

Adding new information can be performed explicitly or implicitly. Explicit addition entails a module creating a new instance of a particular chunk-type, whereas implicit addition entails collecting chunks from module buffers at the conclusion of an ACT-R cycle. Chunks that are referenced by the collected chunks and which do not currently exist in declarative memory are then created automatically.

To merge chunks, candidates must have the same values for the slots that they share. If a chunk-type's slots have been extended at any point prior to a merge, this can cause issues during the merge process since a statically extended chunk will not have a value for an extended slot unless the slot value has been explicitly set, increasing the chance of a chunk mismatch. Non-statically extended chunks, however, will have default values set for extended slots, so the chance of a chunk mismatch occurring is reduced.

Declarative memory topology appears to loosely reflect the external environment that an ACT-R model is situated in: frequency of chunk presentation is only considered when merging chunks and does not affect the topology of declarative memory. In addition, all chunks remaining in module buffers are always completely learned at once, so presentation frequency is disregarded during this operation. The only impact upon topological structure appears to be caused by the order of chunk presentation, since it is only chunks that are present in module buffers at the conclusion of a cycle that are assimilated into declarative memory, and module buffers may only store one chunk at any time.

Procedural System

The procedural system is composed of the *procedural*, *utility* and *production-compilation* modules. It is responsible for producing ACT-R's rational behaviour by maintaining a set of production rules that produce optimal[4] chunks in response to input chunks. Since this paper is concerned with LTM structure, the only modules that will be considered further in the procedural system are the procedural and production-compilation modules. The procedural module stores productions and the production-compilation module is concerned with creating new productions from existing ones.

Adding productions can occur at compile and run-time; if there are no productions specified by a modeller at compile-time, production compilation can not occur at run-time since there are no pre-existing productions to compile. Productions have no topological organisation within the procedural module and adding a production whose name already exists in procedural memory causes the old production to be replaced by the new one.

To modify productions, the production-compilation module collapses two distinct productions into one. Therefore,

[4]Equivalent to a production's utility.

after multiple production compilations, an ACT-R model can produce a sequence of actions without considering intermediate inputs as it did previously. For example, to produce the answer to a mathematical operation such as "24 + 57", a model may simply write "81" in response to this input after production compilation, rather than using an algorithm that divides the numbers into units and adding them together. If a newly compiled production, P', is semantically equivalent to a production that has not been created through production compilation, P, then P' is discarded. If P' is semantically equivalent to a production that has been created through production compilation, $P*$, and utility learning is enabled in ACT-R, P' is not added but the utility of $P*$ is updated.

Productions are only compiled if they meet a set of conditions. Most of these check syntactic aspects of production rules so that they can be feasibly combined within the computational constraints of ACT-R's architecture and so are not discussed here. Those of interest are: productions must have been executed in sequence, and the time between the relevant productions being activated must be less than the threshold time specified. These conditions, in conjunction with the implicit constraint that only two productions can be compiled at a time, mean that production compilation is *incremental* and enforces *temporal contiguity*.

CHREST

CHREST's implementation of LTM contains one data structure comprising a hierarchical discrimination network that indexes a pool of *nodes* connected by *test-links*. Nodes contain *chunks* (Chase & Simon, 1973), and in combination with test-links, enable LTM to provide similarity functions and act as a retrieval device. CHREST's current implementation divides LTM memory into three modalities: action, auditory and visual. These three modalities are root nodes in LTM; chunks presented to CHREST must have their modality specified so LTM can be organised appropriately.

CHREST stores the entirety of a chunk's information in the chunk's reference. For example: a chunk containing an addition fact such as `<[26][+][6]>` is composed of three *primitives*: `[26]`, `[+]` and `[3]`. Its semantics, "this is an addition fact", are not explicitly represented by the contents of the chunk. Encoding perceptual information this way makes it possible to act on a pattern rapidly (Lane & Gobet, 2011).

Learning in CHREST is *incremental* and *on-line*: external information is learned in discrete steps during the model's interaction with its external environment. For example, if the addition fact above were presented, each individual primitive must be committed to LTM in discrete operations (see *Discrimination & Familiarisation* section below for an explanation) before the concatenation thereof can be committed to LTM[5]. This incremental, on-line learning enables CHREST to pick up the statistical distribution of the environment it is situated in naturally, a feature that is critical for simulating expert behaviour and, particularly, the acquisition of language (Jones, Gobet, & Pine, 2007).

Four procedures are used to add or modify LTM information and are considered in detail: discrimination, familiarisation, node linkage and template creation/modification. If any of these procedures are being performed, subsequent requests are blocked. The times taken for LTM to complete each of these procedures are distinct and can be set by modellers at run-time. However, times for discrimination and familiarisation are considered to be part of the architecture since they have been validated independently by empirical research (Feigenbaum & Simon, 1984; Gobet et al., 2001).

Discrimination & Familiarisation

Discrimination and familiarisation are the procedures by which CHREST adds new nodes to LTM or modifies existing ones, respectively. Therefore, discrimination increases the total number of nodes in LTM and familiarisation increases the size of individual nodes in LTM. These procedures rely upon chunks presented to CHREST having a *finished* property set that indicates a complete unit of information.

Discrimination occurs when any of the following conditions are true for a pattern presented to LTM, P, and a chunk retrieved from LTM after P has been presented, C. Tests are applied in the order specified and are cumulative:

- C is a root node for a modality
- C's *finished* property is:
  - True and:
    - The number of primitives in C isn't equal to the number of primitives in P.
    - P's *finished* property is set to false.
  - False and the number of primitives in P is less than the the number of primitives in C.
- A primitive in P is not contained in C.
- The order of primitives in P is not the same in C.

When discrimination occurs, a new test-link is added from C containing the first mismatched primitive in P. Thus, CHREST's incremental learning is *hard-coded* and uses the least amount of information possible to discriminate between external domain features in keeping with the concept of bounded rationality (Simon, 1955) and expert behaviour in general (Gobet et al., 2001).

Familiarisation appends a new primitive from P to C and occurs if the number of primitives in P is greater than in C and C's *finished* property is set to false. As with discrimination, only one primitive is added to LTM, i.e. the first primitive of P, p, that is not present in C. Note that p must be present in LTM before it can be appended to C.

[5] For a discussion of data supporting this design see Feigenbaum and Simon (1984).

Node Links

Node links give CHREST's LTM a graph flavour; they can exist between nodes that have different descendent paths through LTM. Thus, horizontal and vertical traversal of LTM is possible. *Similarity* links are created without modeller intervention when a user-defined number of duplicate primitives, n, exist in two distinct visual chunks that are both completely committed to LTM and present in visual STM. The latter constraint ensures that links between nodes are based on a spatial or temporal contiguity, preserving an essential property of perceptual chunking (Gobet et al., 2001). Unlike discrimination and familiarisation, the order of primitive occurrence in chunks presented to CHREST does not factor into the creation of similarity links. Note that semantic links are bi-directional too: if a similarity link exists between two LTM nodes N and N', it is possible to retrieve N' from N and vice-versa.

Production links are created at run-time without modeller intervention between a visual node and an action node that are held at the same time in visual and action STM. These links hold a value that can be used, for example, to indicate the utility of a production. For example, in chess, if the visual pattern `<[p g 2][p h 2]>` and the action pattern `<[p g2 g3]>` are held in STM, then a production link can be created, with the visual pattern as a condition and the action pattern as the output. When this production is used, its associated value can be incremented or decremented accordingly to denote the utility of the production to inform action-selection in subsequent situations. Note that the visual pattern and the action pattern can be of arbitrary complexity.

Template Creation and Modification

Templates (Gobet & Simon, 1996) evolve from frequently retrieved LTM nodes, N, that contain a number of *core* primitives in their chunk, c, and a number of *varying* primitives, v, in their chunks that are either descendants of, or have similarity links to, N; values of c and v can be set by the user. If N is converted to a template, CHREST attempts to convert any children of N that can become templates into templates too but not nodes that are linked to using similarity links. When converted into a template, N contains *slots* that can have v primitives swapped in/out quickly. Information in slots can concern locations of objects, types of object or chunks can be (recursively) encoded into template slots. Currently, template generation itself incurs no time cost. However, filling a slot has a default time cost of 250ms and this value is considered to be part of template theory.

Architecture Comparison

Given the descriptions provided in the *ACT-R* and *CHREST* sections above we now outline similarities and differences between the concepts discussed, namely: topological LTM structure, chunk structure, chunk addition/modification and chunk linkage. This section is split in two: the first part discusses similarities between the architectures and the second expounds their differences. This comparison offers insights into what cognitive modellers appear to agree on and should be taken forwards into UTCs, new CCAs or new versions of ACT-R and CHREST, and new ideas that could significantly advance the state-of-the-art for cognitive science.

Similarities

Notably, both ACT-R and CHREST use *chunks*, i.e. aggregated features of the external environment, as their LTM currency. Consequently, it seems sensible to propose that a UTC should also use chunks as its cognitive units. Addition of chunks into LTM in both architectures can be *on-line*, i.e. during the model's interaction with an environment and, potentially, incremental (see the *Declarative Module* section for why incremental learning may not be always implemented in ACT-R). Furthermore, the mechanism that controls *implicit* chunk addition is also similar between ACT-R and CHREST: if a chunk, C, references another chunk C' and C' is not yet learned (present in LTM) then, if C is already present in LTM, both ACT-R and CHREST will attempt to add C' to LTM automatically. However, the likelihood of C' being committed to LTM differs between CHREST and ACT-R: in ACT-R this is guaranteed but is not in CHREST.

With regard to controlling whether or not addition/merging and discrimination/familiarisation occurs when requested, the paired operations of ACT-R and CHREST are more similar than they are different. Addition and discrimination will only add a chunk to LTM if that chunk does not already exist in LTM and to modify or familiarise a chunk the shared information in the chunks to be merged must be the same.

In both architectures, chunks are also capable of having production links created between them. These production links originate and terminate with distinct chunks and each production incorporates a measurement of utility. This would suggest that the concept of productions is something that is agreed upon although, currently, CHREST only supports production links between visual and action chunks whereas this restriction does not appear to exist in ACT-R.

Differences

The salient difference between ACT-R and CHREST is ACT-R's use of a sub-symbolic system and CHREST's non-use of such a system. It may be that cognition, in reality, uses a combination of ACT-R and CHREST's approaches. The topological structure of CHREST tends to represent the statistical distribution of the environment since the order and frequency of external information has a large effect upon how test-links, nodes and links between nodes in LTM are formed. Such a complete reflection of the external environment is missing in the structure of declarative memory of ACT-R. Instead, order and frequency of external information presentation is embodied more in the sub-symbolic aspects of ACT-R's LTM. The crucial idea that stems from this comparison is that it may be the case that whilst frequently encountered information is organised in the "specialised" hierarchical discrimination network implemented by CHREST, infrequently encoun-

tered, "general" knowledge may organised in the less structured network implemented by ACT-R. The exact functionality for memory retrieval would then differ depending upon whether external information is represented in the specialised area of LTM or not (if it is, the sub-symbolic system could be given less precedence and vice-versa). In other words, a CHREST-like structuring of LTM may emerge in human LTM after information has first been assimilated and structured in a manner akin to ACT-R's LTM structuring. This modification of structure could provide a measure of expertise in a particular domain and the resulting architecture may provide a more complete and psychologically valid model of human LTM structure.

To control transposition of generic LTM information into specialised LTM information, one could make use of the sub-symbolic functionality and LTM node meta-data that is already present and maintained in ACT-R. For example, given a certain activation level, a LTM node may then be selected for transposition into specialised LTM. This would entail that nodes are tagged with their modality, a feature currently unsupported by ACT-R, but trivial to implement. Such a mechanism would provide a precise, unambiguous and formal basis for topological structuring of LTM, since sub-symbolic information would control whether information becomes hierarchically structured or not. Used in conjunction with long-term human data regarding learning in a particular domain, this hybrid theory's psychological validity could be determined adequately.

It may also be interesting to combine ACT-R's sub-symbolic system with CHREST so that it directly influences the structure of LTM components such as templates. Chunks with higher activation values could gain precedence for template slots and would therefore be swapped into a slot space before a chunk with a lower activation value. This follows the idea of Baddeley (1990), where frequently encountered chunks are processed and memorised more quickly.

With regard to how ACT-R and CHREST structure chunks, ACT-R is much less restrictive with respect to classifying chunk types than CHREST. In ACT-R, it is possible for a user to define chunk types at will, allowing super or sub-classes of chunk types to be created freely. Conversely, CHREST's chunk types are governed by the *modalities* of their constituent primitives and are essentially constricted by the input interface used to generate a chunk. It would seem plausible to suggest that human cognition can make use of both strategies, i.e. whilst certain information is encoded as being of a particular modality: visual, auditory etc., higher-order classifications can also be applied that may be entirely novel. For example, whilst one may construct a visual chunk containing a mathematical formula, we could classify that formula as being an instance of a *mathematical-operation* chunk type. More specifically, the formula may be an instance of an *addition* chunk type (a sub-class of the *mathematical-operation* chunk type). If a CCA were produced that is capable of organising LTM into specialised and general memory (see previous paragraph), these chunk-types may serve to help organise general knowledge topologically, facilitating a directed search and reducing reliance upon *brute-force* retrieval methods that may cause the *utility problem* noted in investigations using large LTMs in ACT-R (Kennedy & Trafton, 2006; Rodgers, Douglass, & Ball, 2009).

When adding information to chunks, ACT-R and CHREST differ in how the existence of this new information is checked. When performing addition, ACT-R only checks to see if the reference for a chunk exists in order to determine whether a chunk should be added. CHREST checks that the chunk's information does not already exist in the order specified. Therefore, CHREST is more concerned with presentation order than ACT-R given its goal of modelling expertise development. It may be that this checking behaviour could be toggled if the a retrieved chunk is part of specialised LTM or not.

Another interesting difference between chunk structure in ACT-R and CHREST relates to where the information in a chunk is contained. In both ACT-R and CHREST it appears that every piece of information used in LTM needs to be internalised as a chunk before it can be used. However, from what we have been able to ascertain research undertaken thus far, CHREST may encode redundant information by duplicating chunks. For example, if a LTM node encodes a chunk, `<[26]>` then, if this chunk is present in another chunk `<[26][+][3]>`, the chunk containing the single primitive is not referenced in the chunk that contains multiple primitives. Instead, the single chunk is duplicated. According to ACT-R however, a chunk is stored in one location in LTM and that location is referenced whenever the chunk is used in another chunk, removing redundancy. Determining which implementation is psychologically valid is an interesting research question and could help further the current state-of-the-art.

With regard to productions, ACT-R and CHREST differ in a number of ways. First, production creation in CHREST is automatic (no modeller intervention required) and entirely novel productions can be created at run-time. Conversely, ACT-R requires modellers to specify an initial set of productions at compile-time and can only compile these pre-defined productions at run-time. Second, production granularity differs with a single production firing in a wide range of situations in ACT-R, so long as its conditions are met and productions are optimised by tuning their parameters. In contrast, productions in CHREST are more akin to micro-productions since their conditions tend to be specific and their range of application limited. In this sense, it seems that ACT-R is able to produce, in certain cases, abstract productions that may be used when new situations occur (where particular productions do not apply) but appear to be similar to previous situations. Currently, CHREST is not capable of creating productions in this way and may therefore benefit from a consideration of how ACT-R achieves such functionality. Unifying these processes may produce a clearer picture of production generation for a UTC. Finally, ACT-R's production rules allow for sequences of actions to be compiled and performed in one

step whereas in CHREST, sequences of actions are possible but not the compilation of two productions. This is an area of potential development and also raises an interesting observation/question: if experts can perform sequences of moves, are these learned production sequences ever revised? If so, is the whole sequence revised after its execution or does an expert consider each subsequent production in the sequence whilst the sequence is being performed?

Conclusions and Future Work

Our intention in this paper has primarily been to create a centralised repository of information regarding the LTM structuring approaches used by ACT-R and CHREST. In the long-term, we hope that this will provide an efficient resource for cognitive modellers to decide between ACT-R and CHREST and to tailor their experiments appropriately given the information discussed. In addition, we also attempted to outline where these CCAs overlap/differ in order to both facilitate agreement on underlying processes of LTM structuring for a UTC, and highlight questions that need to be answered before other UTC concepts can be formalised. In the short-term, we hope this this work will foster a constructive dialogue and exchange of ideas between ACT-R and CHREST's development teams/user-bases, who are currently disconnected.

For those interested in using ACT-R or CHREST, the crucial consideration depends upon how much flexibility with regard to LTM operations a domain-modeller requires. ACT-R contains less hard-coded functionality (it is possible to have an ACT-R model learn 30 chunks at once, for example), whereas CHREST hard codes structural functionality that imposes adherence of the architecture to the principle of *bounded-rationality*. Whilst increased flexibility provides the ability to implement and test multiple theories of cognition, it also increases the programming overhead for domain-modellers since LTM functions will also need to be scheduled in addition to the creation of an input/output interface etc.

In future work we intend to take the ideas delineated in the *Differences* sub-section of the *Architecture Comparison* and expand upon them. Of particular interest to us is the idea that general LTM knowledge (organised topologically in a graph-like manner implemented by ACT-R) may become specialised (topological structuring becomes hierarchical as in CHREST) when certain sub-symbolic conditions are met.

References

Anderson, J. R. (2007). *How can the human mind occur in the physical universe?* Oxford University Press.

Baddeley, A. D. (1990). *Human memory: Theory and practice*. Boston: Allyn & Bacon.

Bothell, D. (n.d.). *ACT-R 6.0 reference manual - working draft.* Retrieved from `http://act-r.psy.cmu.edu/actr6/reference-manual.pdf`

Chase, W. G., & Simon, H. A. (1973). The mind's eye in chess. In W. G. Chase (Ed.), *Visual information processing* (pp. 215–281). New York: Academic Press.

Feigenbaum, E. A., & Simon, H. A. (1984). EPAM-like models of recognition and learning. *Cognitive Science*, *8*, 305–336.

Gobet, F., & Lane, P. C. R. (2010). The CHREST architecture of cognition: The role of perception in general intelligence. In E. Baum, M. Hutter, & E. Kitzelmann (Eds.), *Proceedings of the 3rd conference on artificial general intelligence* (Vol. 10, pp. 7–12).

Gobet, F., Lane, P. C. R., Croker, S. J., Cheng, P. C.-H., Jones, G., Oliver, I., & Pine, J. M. (2001). Chunking mechanisms in human learning. *Trends in Cognitive Sciences*, *5*, 236–243.

Gobet, F., & Simon, H. A. (1996). Templates in chess memory: A mechanism for recalling several boards. *Cognitive Psychology*, *31*, 1–40.

Jilk, D. J., Lebiere, C., O'Reilly, R. C., & Anderson, J. R. (2008). SAL: an explicitly pluralistic cognitive architecture. *Journal of Experimental and Theoretical Artificial Intelligence*, *20*(3), 197-218.

Jones, G. A., Gobet, F., & Pine, J. M. (2007). Linking working memory and long-term memory: A computational model of the learning of new words. *Developmental Science*, *10*, 853–873.

Kennedy, W. G., & Trafton, J. G. (2006). Long-term symbolic learning in SOAR and ACT-R. In D. Fum, F. D. Missier, & A. Stocco (Eds.), *Proceedings of the 7th international conference on cognitive modeling* (p. 166-171).

Lane, P. C. R., & Gobet, F. (2011). Perception in chess and beyond: Commentary on Linhares and Freitas (2010). *New Ideas in Psychology*, *29*, 156–61.

Lende, L. K., & Taatgen, N. (2012). Modeling representational shifts in learning the number line. In N. Rußwinkel, U. Drewitz, & H. van Rijn (Eds.), *Proceedings of the 11th international conference on cognitive modeling* (p. 175-180).

Martin, M. K., Gonzalez, C., & Lebiere, C. (2004). Learning to make decisions in dynamic environments: ACT-R plays the beer game. In M. Lovett, C. Schunn, C. Lebiere, & P. Munro (Eds.), *Proceedings of the 6th international conference on cognitive modeling* (Vol. 420, p. 178183).

Newell, A. (1990). *Unified theories of cognition*. Cambridge, MA: Harvard University Press.

Oliva, J., Serrano, J. I., del Castillo, M. D., & Ángel Iglesias. (2010). Cognitive modeling of the acquisition of a highly inflected verbal system. In D. D. Salvucci & G. Gunzelmann (Eds.), *Proceedings of the 10th international conference on cognitive modeling* (p. 181-186).

Rodgers, S. M., Douglass, S. A., & Ball, J. (2009). Large declarative memories in ACT-R. In A. Howes, D. Peebles, & R. P. Cooper (Eds.), *Proceedings of the 9th international conference on cognitive modeling* (p. 222-228).

Simon, H. A. (1955). A behavioral model of rational choice. *The Quarterly Journal of Economics*, *69*, 99–118.