Computational and Explanatory Power of Cognitive Architectures: The Case of ACT-R

Holger Schultheis (schulth@sfbtr8.uni-bremen.de)
SFB/TR 8 Spatial Cognition, Universität Bremen, Enrique-Schmidt-Str. 5, 28359 Bremen, Germany

Abstract

Cognitive architectures constitute a generally preferable approach to create computational accounts of human cognition. Yet, cognitive architectures are also hard to assess. Following up on and extending the work of Cooper (2007), we further assess the popular cognitive architecture ACT-R in this paper. It turns out that ACT-R fares worse than one may expect both regarding the scope of empirical effects it has been shown to account for and regarding its explanatory power.

Keywords: cognitive architectures; Lakatos; Turing machines; counter machines; ACT-R.

Cognitive Architectures and Their Assessment

One approach to building computational models is to develop the model as part of a *cognitive architecture*. Cognitive architectures can be characterized as implemented theories of the fixed mechanisms and structures that underlie human cognition. As such cognitive architectures strive to offer a framework in which all of human cognition can be modeled. Building on the common mechanisms provided by the architecture, computational models for particular domains or tasks can be created by adding task and domain specific information to the architecture (cf. Lehman, Laird, & Rosenbloom, 1998). Thus, a cognitive model developed in the scope of a cognitive architecture can be viewed as consisting both of the architectural mechanisms and the task / domain specific information (i.e., content) added to the architecture.

Employing cognitive architectures for modeling human cognition has the advantage that otherwise isolated and fragmentary accounts of human cognition can be integrated to ultimately (hopefully) yield an account of human cognition as a whole (Newell, 1990). In this sense, building models in cognitive architectures is preferable to building isolated models. Obviously, this advantage of cognitive architecture will only hold, if the employed architecture is a good approximation of the general mechanisms and structures that underlie human cognition. To not jeopardize the aim of arriving at a veridical account of all human cognition, the quality of the cognitive architecture needs to be assessed and possibly improved by changing the architecture.

As Cooper (2007) points out, assessing cognitive architectures is less straightforward than assessing isolated models. Whereas isolated models lend themselves naturally to Popperian falsification, cognitive architectures do not. Against this background, already Newell (1990) argued that the development of cognitive architectures should be guided not by Popperian falsification but by criteria as arising from the theory put forth by Lakatos (1970). Following this suggestion and further supporting it, Cooper (2007) employs Lakatosian criteria to assess the two architectures Soar (Newell, 1990) and ACT-R (Anderson, 2007).

In this paper we bring to the foreground further criteria for assessing the merit of cognitive architectures. Moreover, we combine these additional criteria with the Lakatosian criteria described in Cooper (2007) to continue the assessment of the cognitive architecture ACT-R. To do this, we first briefly describe the notions and criteria relevant for the assessment. Subsequently, we assess ACT-R regarding these criteria. This comprises (a) describing those aspects of ACT-R which are most relevant for the presented assessment and (b) conducting formal and literature analyses to assess ACT-R's standing with respect to the considered criteria. Finally, we close with some implications the assessment's results have for (the future development of) ACT-R.

Assessment Criteria

Lakatosian Criteria

According to Lakatos (1970) scientific development occurs in the scope of so called research programs. Roughly speaking, each such research program comprises both a *hard core* and a *protective belt*. The hard core consists of all those assumptions which are central to the program, that is, giving them up would mean to give up the research program. In contrast, the protective belt is made up of assumptions and hypotheses of a more peripheral nature, that is, assumptions which may help to further specify aspects of the research program, but to which the research program is not irrevocably committed.

Research programs generally develop by (empirically) testing predictions derived from the hard core and the protective belt. If the predictions are confirmed, this supports the research program. If the predictions are refuted, this may lead to a change of the protective belt (i.e., some peripheral assumptions) of the research program. Depending on the consequences of the change of the protective belt, Lakatos (1970) calls a research program *theoretically progressive* or not. A research program is theoretically progressive if and only if the change of the assumptions increases the empirical content of the research program, that is, allows the research program to account for more empirical phenomena than before the change. Importantly, research programs which are not theoretically progressive are not scientific but only pseudo-scientific. Lakatos (1970) further categorizes research programs as to whether they are *empirically progressive* or not. If and only if a research program's predictions are empirically confirmed, it is empirically progressive.

Sticking to Lakatos' terminology, cognitive architectures are research programs. Accordingly, one can use the notions

of theoretical and empirical progressiveness to assess cognitive architectures. Consequently, following up on and adding to the work of Cooper (2007), we more closely consider ACT-R's theoretical progressiveness in this contribution.

To Can and Cannot

The above outlined Lakatosian criteria stress the ability of a cognitive architecture to account for empirical findings[1]. What a cognitive architecture can account for is, however, only one part of an architecture's quality. As Roberts and Pashler (2000) remark, it is equally important to decide on the quality of a given architecture to know what the architecture cannot account for. Neglecting the "cannot" aspect is a serious problem, because a cognitive architecture is intended to provide the basis to explain human behavior and not arbitrary behavior.

To illustrate the problem, consider a certain architecture S which allows to model a certain empirically found effect f. Let us assume that f is a reaction time difference between two experimental conditions A and B such that reaction times are longer in A. Assume further that $\bar{f}$ is the hypothetical (i.e., not observed) effect that reaction times are longer in condition B. An interesting question now is whether S also allows to model $\bar{f}$. If S allows modeling $\bar{f}$, S accounts for both the empirically found effect and its opposite.

Given such a situation, the explanatory value of S is called into question. S is a cognitive architecture and should, thus, realize the mechanisms and structure underlying human cognition. S's ability to account for both f and $\bar{f}$ undermines its assumed cognitive plausibility, because humans do only behave according to f but not according to $\bar{f}$. If the structure and mechanisms of the human mind constrains human cognition and behavior to f, a cognitive architecture which allows modeling $\bar{f}$ is erring with respect to at least some part of the structure and mechanisms underlying human cognition.

Thus, to fully judge the quality of a cognitive architecture, it is equally important to know what the architecture cannot account for as it is to know what the architecture can account for. Ideally, the architectural mechanisms and structure constitute a framework which constrains the content that can be added to it such that the set of all models possible in the architecture accounts for and only for all phenomena empirically observable in human cognition and behavior (cf. also Taatgen, 2003).

In line with its importance and in addition to theoretical progressiveness, the question what can and cannot be modeled in ACT-R is one major point of inquiry in the subsequent assessment of ACT-R.

[1] Strictly speaking, cognitive architectures per se do not account directly for any empirical findings. Only the models which can be build in an architecture can account for empirical phenomena. However, to ease the subsequent exposition we will talk of architectures that account for findings instead of using the more cumbersome wording of architectures that allow building models which account for empirical findings.

ACT-R

ACT-R (see Anderson, 2007; Anderson et al., 2004) consists of several components which are called modules. One of these modules, called the production module, stores and executes a set of productions. Each production specifies under which conditions it is applicable. If the current state of the ACT-R system satisfies a production's conditions, the production can be executed which will lead to a change of the state of the system. Additional modules of ACT-R include the declarative module (storing declarative knowledge in the form of proposition-like pieces of knowledge called *chunks*), the goal module (managing the current goal), and several perceptual motor modules (realizing ACT-R's interaction with the environment). Each of these modules is interfaced to the overall system by a buffer. The working of the procedural module draws heavily on these buffers. Production conditions and effects are specified nearly exclusively in terms of buffer content. The productions conditions are checked against the buffers' content and production application will normally change the content of one or more buffers.

Regarding the Lakatosian criteria of architecture assessment mentioned above it is interesting to what extent one can distinguish the hard core and the protective belt realized by ACT-R. As Cooper (2007) remarks, although the developers of ACT-R have never explicitly used Lakatosian terminology to draw such a distinction, such a distinction suggests itself from the descriptions of the notions underlying ACT-R's development. For example, Anderson (1976, pp. 114) proposes several "preconceived notions" which constitute the skeleton of ACT-R. These preconceived notions, such as to distinguish between and to employ both procedural and declarative knowledge, constitute the hard core of ACT-R and have remained unchanged since their proposal in 1976. All aspects of ACT-R other than the preconceived notions can be viewed as constituting the protective belt. For instance, the formulas and mechanisms used to select one of several productions or one of several pieces of declarative knowledge are part of the peripheral assumptions.

This protective belt of ACT-R is largely parametrizable. Using the parameters the architecture provides one can determine both which of the peripheral assumptions to employ (e.g., whether to use certain formulas to determine which production to select) and how the selected peripheral assumptions behave. Since ACT-R has been first proposed by Anderson (1976), its protective belt has changed considerably. In its current version (6.0 [r723], see Bothell, 2009) which we consider here, ACT-R has about 50 parameters. Only for few of them general recommendations of how to set them exist (Anderson et al., 2004).

The Cannot in ACT-R

As a cognitive architecture, ACT-R constitutes a computational framework for building cognitive models. Due to this computational nature one manifest starting point to investigate what ACT-R cannot do is to ask which subset of the set

of all computable functions cannot be realized in ACT-R. As it turns out, there are no functions which can be computed, in principle, but not in ACT-R. In the following, we prove this constructively by presenting two particular ACT-R models[2].

Universal Turing Machine A Turing machine is a computing machine which was introduced by Turing (1936). A Turing machine consists of an infinite tape partitioned into cells and a control unit moving over the tape. Each cell contains a single symbol and each machine can deal only with a finite, predefined set of symbols. The control unit can read and write on the tape one cell at a time and can move from one cell to one of the two neighboring cell. At every point in time the Turing machine is in one of a finite number of states. Depending on the machine's current state and what is read from the cell currently in focus, the machine will write to the cell in focus and / or move to an adjacent cell.

Although quite simple in their setup, Turing machines have been found to be able to compute a wide range of functions (see Minsky, 1967, for an in-depth treatment of Turing machines and several example machines). More precisely, it is generally assumed—though unproven—that the set of functions which can be computed by Turing machines is identical to the set of all computable functions. What is more, certain Turing machines, called universal Turing machines, are able to emulate the working of any other Turing machine, that is, universal Turing machines can compute anything that Turing machines in general can compute. Put differently, universal Turing machines are computing machines which can compute all computable functions. In the remainder of this section we describe an ACT-R model which emulates a universal Turing machine. The chosen machine is a machine with 4 states and 6 symbols which has been proposed by Rogozhin (1996).

To emulate the chosen machine, the tape of the machine is realized as the content of the declarative module. Each cell on the tape is represented by a chunk in declarative memory. Such a chunk *c* essentially stores (a) the symbol contained in the cell *c* represents, (b) the chunk which represents the cell which would be to the right of the cell represented by *c* on the tape, and (c) the chunk which represents the cell which would be to the left of the cell represented by *c* on the tape.

The goal buffer contains the chunk representing the cell that is currently in focus. In addition to the cell information the goal buffer also stores the current state of the machine.

The reading and writing of information onto the tape as well as the movement of the control unit is realized by productions. Basically, four types of productions are employed to realize the operations of the control unit:

- *update*: Depending on the current state of the machine and the symbol in the current cell (i.e., the corresponding symbol stored in the goal buffer), this type of production writes a symbol into the current cell (i.e., updates the corresponding slot in the goal buffer).
- *prepare transition*: As described above, the combination of a state and symbol also affords a move of the control unit. This type of production prepares such a move. By drawing on the information about the neighboring cells given in the currently focused-on cell, the production requests the retrieval of the appropriate chunk (i.e., the chunk representing the cell to move to).
- *get next*: The "get next" type of production is applicable whenever a chunk representing a cell on the tape is available in the retrieval buffer. The main purpose of this production type is to modify the cell representation in the retrieval buffer such that it can serve as the representation of the current cell in the goal buffer. This preparation comprises basically two things. First, the machine's state as resulting from the previously encountered state-symbol combination is stored in the appropriate slot of the chunk in the retrieval buffer. Second, the chunk currently in the goal buffer is stored as either the right or left neighbor of the cell represented by the chunk in the retrieval buffer. If the control unit has "moved" to the left, the chunk in the goal buffer is stored as the right neighbor and vice versa.
- *do transition*: This type of production replaces the chunk currently in the goal buffer with the chunk currently in the retrieval buffer.

These four types of productions when being executed in the sequence in which they were described constitute one elementary operation of a Turing machine: Read a symbol and then, based on the combination of current state and the read symbol, write a symbol, update the state and move to the next cell. Since the movement direction, the state to change to, and the symbol to be written depend on the previous state and the read symbol, for each possible state-symbol combination these four productions have to be slightly different. Consequently, to emulate the universal Turing machine in question, our model employs a variation of this 4-tupel of productions for each of the 24 possible state-symbol combinations.

Representing the tape by declarative memory and the working of the control unit by productions as described, allows to emulate the universal Turing machine by running the model in ACT-R. The only thing one has to do to emulate the machine computing a certain function is to provide the initial tape configuration as chunks in declarative memory and to set the initial focus to the appropriate cell of the initial tape configuration. We have successfully emulated several Turing machines using this approach. For these model runs we enabled sub-symbolic processing in ACT-R and set the latency factor parameter to 0.1. All other parameters of ACT-R were left at their default values as described in Bothell (2009).

Consequently, as the presented model runs completely in ACT-R, ACT-R allows to emulate a universal Turing machine. This shows that there is no computable function which cannot be computed in ACT-R. Moreover, the model we describe next demonstrates that this is not the only way to realize universal computation in ACT-R.

[2]Model code is available from http://www.cosy.informatik.uni-bremen.de/staff/schultheis/ICCM09-models/

Universal Counter Machine A second class of computing machines is called counter machines. A counter machine comprises a finite number of registers and can interpret a finite set of instructions. The registers store integer values and can be tested and manipulated by the instructions which are part of the instruction set of the machine. To compute some function f, a counter machine has to be equipped with an initial set of values in its registers and a program, that is, a sequence of instructions from the machine's instruction set. The machine will execute the program and once the end of the program is reached, the result of the computation will be available in one (or more) of the registers.

Minsky (1967, pp. 255) has proven that a counter machine employing only three instructions and two registers can compute any computable function. The required instructions are INC(r_i) (add 1 to register r_i and go to the next instruction), JZDEC(r_i, n) (if $r_i = 0$ go to instruction n, otherwise subtract 1 from r_i and go to the next instruction), and GO(n) (go to instruction n). Since this counter machine is universal, for any computable function f there exists a program (i.e., a sequence of instructions) and an initial value for both registers such that the counter machine computes f.

As in the case of Turing machines, it is possible to emulate computation using counter machines by devising appropriate ACT-R models. To show this, it suffices to explain how an ACT-R model can realize (a) the two registers, (b) the three instructions, and (c) the sequence of instructions. In our model, the two registers are realized as slots in a chunk, where this chunk remains in the goal buffer for the complete model run. The instructions are realized as productions. To control the sequence of instructions a third slot in the chunk in the goal buffer stores a label. This label is tested in the condition of the productions such that only the production corresponding to the current label is applicable. Against this background the three types of instructions outlined above can then be transcribed by productions as follows:

- INC(r_i): This instruction is realized by reading the current value of r_i from the goal chunk, adding 1 to that value by using the `!bind!` statement of ACT-R, and storing the resulting value again in the goal chunk.

- GO(n): To effect such a GO statement, a production needs only to change the label in the goal chunk such that it corresponds to instruction n.

- JZDEC(r_i, n): Two productions are necessary to transcribe this instruction. Both productions test the content of r_i using the `!eval!` statement of ACT-R. The first production is only applicable if $r_i = 0$ holds and essentially works as the production mimicking the GO instruction. The second production is only applicable if $r_i > 0$ holds and subtracts 1 from r_i analogous to the workings of the INC instruction.

Importantly, these methods for transcribing a program of the universal counter machine as an ACT-R model, are not program specific. Put differently, any program formulated for the universal counter machine can be transcribed as an ACT-R model. Consequently, the universal counter machine can be completely emulated in ACT-R. To illustrate the emulation of the counter machine, we implemented a model which computes the sum of two numbers. The parameter settings for this model are identical to those used in the Turing machine model. By appropriately initializing the first register, running the model computes the sum of the two numbers and encodes the result as a number in the first register.

The possibility to emulate a universal counter machine in ACT-R provides additional evidence that there is no computable function that cannot be realized in ACT-R. Although this second evidence may seem unessential, as explained in the next section, the fact that universal computation can be realized in ACT-R in more than one way is of relevance for assessing the architecture.

Summary and Discussion Both models presented above paint a clear picture of which functions cannot be realized in ACT-R: There is simply no computable function that cannot be computed using ACT-R. That is, ACT-R does not seem to fulfill the requirement to constrain the models that can be built in it too well. Consequently, at least regarding the "cannot" criterion ACT-R fares poorly.

One may be inclined to object to this conclusion or the way it was brought about. Therefore, we list and discuss several possible objections in the remainder of this section.

First, one may argue that the fact that ACT-R is Turing-complete is neither new nor problematic. Regarding originality, Anderson (1976, pp. 140) already presented the sketch of a proof of ACT-R's Turing completeness. However, the proof presented in Anderson (1976) refers to the initial version of the cognitive architecture. Over the past 30 years the overall setup of the architecture has changed considerably. In particular, certain changes (see e.g., Anderson & Lebiere, 1998, p. 440) were explicitly implemented to reduce the computational power of ACT-R. Thus, the Turing completeness of ACT-R in its current version could not be derived from the 1976 proof, but had to be newly established.

Yet, Turing completeness of ACT-R (or any cognitive architecture) may not be considered a problem. In proposing the physical symbol system hypothesis Newell (1980) argued that any system able to realize human-level intelligence necessarily needs to be Turing-complete. Against this background, it may not be immediately obvious why the above described models constitute problems for ACT-R. The problem is that it is unclear and dubitable that the presented models realize Turing completeness appropriately. As the two models indicate, Turing completeness can be realized in several ways. When using universal computing machines to achieve results in computation theory it may not be crucial which of all possible realizations of universal computation one employs. For a cognitive architecture such as ACT-R, however, the way universal computation is achieved is essential. Striving to constitute a theory of human cognition as a whole, ACT-R must realize Turing completeness in the same way

as Turing completeness is achieved in the human cognitive system. Among other things, this requires that the timing behavior of the architecture and of human cognition match closely. Thus, computing any function f in ACT-R should take about the same time as human cognition requires to compute f. Put differently, to live up to its aim of being a satisfactory cognitive architecture, ACT-R should not allow to compute f considerably faster or slower than the human cognitive system computes f. This is not the case, since, as the two models show, ACT-R can be made to compute any f in a wide range of times. Different universal machines diverge considerably with respect to the time they need to compute any f (e.g., Woods & Neary, 2009). The two models prove that there is a wide range of universal machines which can be realized in ACT-R. Not only does ACT-R allow implementing different types of universal computing devices (i.e., Turing machines and counter machines), but also for each of these types numerous instances can be realized. For example, one could implement a universal Turing machine with different states, symbols, and transition rules (Rogozhin, 1996). Likewise, universal counter machines with different instruction sets and / or more registers (Minsky, 1967) can be built in ACT-R analogously to the second model described above. Thus, ACT-R allows to realize any function with a wide range of times. In the worst case, it may even be possible that any function can be realized in arbitrary time in ACT-R. Regardless whether this is the case, it seems clear that their is too few which ACT-R cannot do, to consider ACT-R as satisfactorily constraining what can be implemented in it.

A second objection that may be raised concerns the compliancy of the models, that is, the extent to which the models are formulated in keeping with the spirit of the architecture (Young, 2003). Perhaps one may want to argue that some of the model's components are violating one or more theoretical stances implicitly being part of the architecture. One difficulty with such an argument is that there is no clear and explicit definition of what type of model components do and which type of model components do not keep with the spirit of ACT-R. In addition, for constituting a satisfactory account of the fixed mechanisms and structures underlying human cognition, it should be the architecture itself and not some code of how to use the architecture that constrains what models one can build in the architecture. Thus, an objection in terms of compliancy fails to address the core issue brought up by the above presented models and considerations.

In summary, the presented analyses indicates that what cannot be done in ACT-R is considerably less than desirable. Multiple realizability of universal computation on several different time scales leaves too much room for implementing behavior which ACT-R should not allow to be implemented. As a result, ACT-R's ability to meet the "cannot" criterion is, to say the least, debatable.

The Can in ACT-R

After having considered what ACT-R cannot do, we now turn to the question what ACT-R can do. A first answer to this question directly derives from the models presented above: ACT-R allows to compute every computable function. However, this is, as also mentioned above, only a partially satisfactory answer. To fully judge ACT-R's "can" ability, it is important to more closely consider whether ACT-R allows to compute these functions as the human cognitive system computes them (e.g., regarding timing). Essentially this amounts to examine for which tasks and domains of cognition ACT-R models can be built that closely mimic human behavior and cognition. In Lakatosian terms, it is necessary to examine the empirical content of ACT-R.

Judging from the plethora of publications on ACT-R (models) listed on the ACT-R web site one would expect that ACT-R does well with respect to this empirical content criterion. To verify this impression of ACT-R's empirical content, we reviewed all papers presenting ACT-R models which were listed on the ACT-R site as being published either 2007 or 2008. These two years were chosen because they presumably represent the current state of the art in ACT-R modeling.

Overall 35 papers presenting models accounting for various aspects of human cognition are available. This is an impressive number which seems to indicate the large empirical content encompassed by ACT-R. On closer inspection, however, it turns out that the empirical content of the current ACT-R version is (a) unclear and (b) probably less than suggested by the number of presented models. The reason for this is the way the ACT-R community proceeds with the change of parts of the protective belt of ACT-R—both across and within different ACT-R versions.

Each change in version is accompanied by a change of at least some of the peripheral assumptions in ACT-R. For example, from ACT-R 4 to ACT-R 5 the goal stack was replaced by the goal buffer and from ACT-R 5 to ACT-R 6 the formula for computing production utility was considerably modified. Although there is, of course, nothing wrong with such changes per se, for each of these changes it is mostly unclear whether they are theoretically progressive, that is, whether they increase the empirical content of ACT-R. To show that the current ACT-R version's content is increased compared to its predecessors would require to prove (by reimplementation in the current version) that empirical phenomena accounted for by older ACT-R versions can still be accounted for by the current version. Such reimplementation is rarely done. On the contrary, even 2008 published modeling work is partly conducted in ACT-R 4 (e.g., Altmann & Gray, 2008) and ACT-R 5 (e.g., Gunzelmann & Gluck, 2008).

Furthermore, even for single ACT-R versions, the empirical content is unclear. There are mainly two reasons for that: First, by appropriately setting particular parameters several peripheral assumptions can be and are switched on or off at will. For instance, some models employ base-level learning or production learning while others do employ neither. Second, it is not unusual for ACT-R modeling work to modify or extend the protective belt. Of the 35 modeling paper reviewed, 17 considerably changed the protective belt, for ex-

ample, by changing existing modules (e.g., Maanen & Rijn, 2007) or adding new modules (e.g., Juvina & Taatgen, 2007).

This frequent change of the protective belt across and within ACT-R versions renders it difficult to judge the empirical content of the current version of ACT-R. Whether all the modeling work employing differing protective belts is reconcilable is an open question. Due to this problem, it is not clear whether ACT-R is theoretically progressive. But even if it is, ACT-R's empirical content remains to be determined.

Conclusion

In this paper we picked up on and extended the methodology proposed by Cooper (2007) to assess the merit of cognitive architectures. We applied the methodology to one of the most commonly employed cognitive architectures, ACT-R. In this assessment, ACT-R fares worse than one may have expected. For one, ACT-R's ability to account for human cognition is less evident than suggested by the available host of modeling papers employing ACT-R. It remains to be investigated to what extent different modeling work can be integrated into ACT-R without varying its peripheral assumptions. Even if ACT-R's ability to account for empirical data turns out to be substantial, the explanatory value of this is called into question by ACT-R's computational power. Since ACT-R in its current version must be assumed to allow computing functions in a lot of ways different from human cognition, it is unclear to what extent ACT-R mirrors and, thus, explains the mechanisms and structure underlying human cognition.

Overall, ACT-R has served and still serves an important function in providing a platform for modeling human cognition. Interesting accounts of various aspects of human cognition have been formalized in ACT-R. Yet, to substantiate ACT-R's status as a cognitive architecture constituting a unified theory of cognition, it is necessary (a) to more closely determine its actual empirical content and (b) to more strongly constrain what ACT-R allows to be implemented.

Acknowledgments

In this paper work done in the project R1–[Image-Space] of the Transregional Collaborative Research Center SFB/TR 8 Spatial Cognition is presented. Funding by the German Research Foundation (DFG) is gratefully acknowledged. We also thank Astrid Kurbjuweit for fruitful discussions and two anonymous reviewers for their constructive comments.

References

Altmann, E. M., & Gray, W. D. (2008). An integrated model of cognitive control in task switching. *Psychological Review*, *115*, 602-639.

Anderson, J. R. (1976). *Language, memory, and thought*. Hillsdale, NJ: Lawrence Erlbaum Associates.

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

Anderson, J. R., Bothell, D., Byrne, M. D., Douglass, S., Lebiere, C., & Qin, Y. (2004). An integrated theory of the mind. *Psychological Review*, *111*(4), 1036 - 1060.

Anderson, J. R., & Lebiere, C. (1998). *The atomic components of thought*. Mahwah, NJ: Lawrence Erlbaum.

Bothell, D. (2009). ACT-R 6.0 reference manual. Retrieved February 5, 2009, from http://act-r.psy.cmu.edu/actr6/reference-manual.pdf. [Computer software manual].

Cooper, R. P. (2007). The role of falsification in the development of cognitive architectures: Insights from a Lakatosian analysis. *Cognitive Science*, *31*, 509-533.

Gunzelmann, G., & Gluck, K. A. (2008). Approaches to modeling the effects of fatigue on cognitive performance. In *Proceedings of the 17th Conference on Behavior Representation in Modeling and Simulation.*

Juvina, I., & Taatgen, N. (2007). Modeling control strategies in the n-back task. In *Proceedings of the 8th International Conference on Cognitive Modeling.*

Lakatos, I. (1970). Falsification and the methodology of scientific research programs. In I. Lakatos & A. Musgrave (Eds.), *Criticism and the growth of knowledge.* Cambridge, UK: Cambridge University Press.

Lehman, J. F., Laird, J., & Rosenbloom, P. (1998). A gentle introduction to Soar, an architecture for human cognition. In S. Sternberg & D. Scarborough (Eds.), *Invitation to Cognitive Science, Volume 4.* Cambridge, MA: MIT Press.

Maanen, L. van, & Rijn, H. van. (2007). An accumulator model of semantic interference. *Cognitive Systems Research*, *8*, 174-181.

Minsky, M. (1967). *Computation – Finite and infinite machines*. Englewood Cliffs, NJ: Prentice Hall.

Newell, A. (1980). Physical symbol systems. *Cognitive Science*, *4*, 135 - 183.

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

Roberts, S., & Pashler, H. (2000). How persuasive is a good fit? A comment on theory testing. *Psychological Review*, *107*, 358 - 367.

Rogozhin, Y. (1996). Small universal turing machines. *Theoretical Computer Science*, *168*, 215-240.

Taatgen, N. (2003). Poppering the Newell Test. *Behavioral and Brain Sciences*, *26*, 621 - 622.

Turing, A. M. (1936). On computable numbers, with an application to the Entscheidungsproblem. *Proceedings of the London Mathematical Society*, *2-42*, 230-265.

Woods, D., & Neary, T. (2009). The complexity of small universal Turing machines: A survey. *Theoretical Computer Science*, *410*, 443-450.

Young, R. M. (2003). Cognitive architectures need compliancy not universality. *Behavioral and Brain Sciences*, *26*, 628.