Cohen, A. M., Ritter, F. E., & Haynes, S. R. (2007). Using reflective learning to master opponent strategy in competitive environments. In *Proceedings of the International Conference on Cognitive Modeling,* 157-162. Oxford, UK: Taylor & Francis/Psychology Press.

# Using Reflective Learning to Master Opponent Strategy in a Competitive Environment

**Mark A. Cohen (mcohen@lhup.edu)**
Department of Business Administration, Computer Science, and Information Technology, Lock Haven University, Lock Haven, PA 17745 USA

**Frank E. Ritter (frank.ritter@psu.edu)**
**Steven R. Haynes (shaynes@ist.psu.edu)**
College of Information Sciences and Technology, The Pennsylvania State University, State College, PA 16802 USA

**Abstract**

Cognitive models of people interacting in competitive environments can be useful, especially in games and simulations. To be successful in such environments, it is necessary to quickly learn the strategy used by the opponent. In addition, as the opponent adjusts its tactics, it is equally important to quickly unlearn opponent strategies that are no longer used. In this paper, we present human performance data from a competitive environment. In addition, a cognitive model that uses reflective learning is introduced and compared to the empirical findings. The model demonstrates that it is possible to simulate learning in an adversarial environment using reflection and provides insight into how such a model can be expanded.

## Introduction

Cognitive models of people interacting in competitive environments can be useful, especially in games and simulations (Jones et al., 1999; Laird, 2001a, 2001b; Ritter et al., 2002). To be successful in such environments, it is necessary to quickly learn the strategy used by the opponent. In addition, as the opponent adjusts its tactics it is equally important to quickly unlearn opponent strategies that are no longer used. The model presented here uses learning by reflection to accomplish this task. This model was created using a high-level tool that produces cognitive models quickly, and with little or no programming. We briefly take up the two important aspects of this project, leaning from reflection and the role of variability in performance.

### Leaning By Reflection

Learning by reflection (or introspection) is one technique that can be used to learn and unlearn an opponent's changing strategies while at the same time preserving the variability in which people learn (e.g. Bass, Baxter, & Ritter, 1995; Cox & Ram, 1999; Ritter & Wallach, 1998).

Learning by reflection is a form of metacognition that allows the model to learn by reflecting on its performance, and adjusting accordingly. When reflection reveals previous actions that were beneficial, the model will be more likely to repeat those same actions in similar situations. However, when reflection reveals poor performance, the actions that lead to that performance are less likely to be repeated. Thus, learning by reflection is a form of reinforcement learning (Russell & Norvig, 2003).

Reflective learning requires that both the cognitive model and its environment are fully observable (Russell & Norvig, 2003). In other words, the model must be able to observe the effects of its actions on the environment and other models.

### Variability

For a model's behavior to be believable in a game or simulation its performance must do more than match average human behavior. Cognitive models must also exhibit the same variability in behavior that a human exhibits. When playing a game or participating in a simulation, variable behavior is a crucial part of the realism that these systems must portray.

Because reflective learning strategies are based on probability, the behaviors they generate are not deterministic. This allows reflective models to exhibit variability in learning and thus performance.

The remainder of this paper describes a study done to measure how quickly participants in a user study learn opponent strategies while performing a competitive task, and a cognitive model that was designed to exhibit similar performance with equal variability.

## Task

Lehman, Laird, and Rosenbloom (1996) in their *A Gentle Introduction to Soar* use baseball repeatedly as an example. This inspired us to implement a simple version of a baseball game to study adversarial problem solving and support people learning Soar. In a broader context, this environment provides an accessible platform for the future study of cognitive models interacting with other agents in a social simulation (Sun, 2006).

Figure 1 shows the basic interface and one of the feedback screens. In this game, participants play the role of the pitcher competing against a series of agent-operated batters. The goal of this game, as in baseball, is to get batters out.

The baseball game described here was written in Java and interacts with the Soar cognitive architecture using the Soar Markup Language (*SML Quick Start Guide*, 2005). The software and instructions on how to use it are available online (acs.ist.psu.edu/herbal).


Figure 1: The Baseball Game Task.

## Rules of the Game

There are two ways to get a batter out in this game: The batter can get three strikes (a strike results when a batter either swings and misses or does not swing at a good pitch), or the batter can hit the ball directly at a fielder who catches the ball.

There are also two ways for a batter to get on base in this game: The batter can get four balls (a ball results when the batter does not swing at a bad pitch), or the batter can hit the ball in a way that prevents the fielders from catching it.

Acting as the pitcher, the participants in this study had a choice of throwing either a fastball or a curveball to the batter. Once they threw a pitch, the batter had a choice of either swinging at the pitch or letting it go by. Both the pitcher and batter are always aware of how many balls and how many strikes the batter has. The rules shown in Table 1 describe how to determine the outcome of each pitch.

Table 1: Determining the outcome of a pitch

| Pitcher | Batter Response | Outcome |
|---|---|---|
| Fastball | Batter swings | Contact is made that may result in either an out (50% of the time) or a hit (50% of the time). |
| Fastball | Batter does not swing | The pitch will result in a strike.[1] |
| Curveball | Batter swings | The pitch will result in a strike.[1] |
| Curveball | Batter does not swing | The pitch will result in a ball.[1] |

Based on the rules described above, the most certain way to get a batter out is to throw a curveball when the pitcher thinks the batter will be swinging and to throw a fastball when the pitcher thinks the batter is not going to swing. Naturally, if the participant can learn what strategy the batter is using then they have a better chance of getting them out.

## Batter Strategies

Each participant faced the same five different batter strategies in the same sequence during play. Strategy changes were determined by the number of consecutive outs that the participant recorded against a given strategy. When a predetermined out threshold was reached, a strategy shift by the batter would take place. The exact sequence of batter strategies and their corresponding out thresholds were defined in a configuration file that was used by the baseball environment, but is unknown to the pitcher. The batter strategies, along with their consecutive out thresholds, are shown in Table 2. The strategies shown here are the ones used in our user study in the order they are listed. However, we do not propose this as the only order, or the best order. The baseball game environment is easily configurable to use other strategies and to present them in any order. This illustrates the baseball task's usefulness for studying the effects of order on learning (Ritter, Nerb, O'Shea, & Lehtinen, 2007).

Table 2: Batter Strategies in the Baseball Environment

| Name | Strategy | Out Threshold |
|---|---|---|
| Hacker | Always swings | 4 |
| Aggressive | Swings at the first pitch and when there are fewer strikes than balls, unless there are three balls and two strikes | 7 |
| Random | Randomly chooses when to swing | 5 |
| Chicken | Never swings | 4 |
| Alternate | Swings if the last pitch was a fastball and does not swing if it is the first pitch or the last pitch was a curve | 7 |

To make it clear exactly how strategy changes took place during the game, an example is provided.

**Strategy Shift Example** The participant begins by facing batters that use the Hacker strategy. Because the consecutive out threshold for this strategy is 4, the participant continues to face batters that use the Hacker strategy until they get 4 consecutive batters out. At this point in time, the strategy shifts to the Aggressive strategy and a new out threshold of 7 is in effect. The Aggressive strategy is then used by the batters until 7 batters are retired consecutively. Game play continues in this fashion until the participant reaches the fifth and final strategy (Alternate). When 7 consecutive Alternate batters are retired by the pitcher the game ends.

[1] *If the batter gets three strikes, then they are out (called a strikeout). If the batter gets four balls, they get a free pass to first base (called a walk).*

### Performance Measure

The participant's ability to learn a particular strategy was measured quantitatively using a measure of pitching efficiency (*PE*). The following formula was used to calculate pitching efficiency:

$$PE = N_s / T_s$$

Where $N_s$ is the number of batters using strategy *s* that were faced by the participant, and $T_s$ is the consecutive out threshold for strategy *s*. A decrease in PE indicates an increase in the efficiency of the pitcher. A value of 1.0 for *PE* indicates the most efficient pitching strategy. For example, if a participant faced 14 Aggressive batters before they could retire 7 in a row, the participant's pitching efficiency would be 14 / 7, or 2.

## Method

Undergraduate Computer Science students at Lock Haven University participated. A total of 10 participants performed the baseball task. Nine of the 10 participants were male.

After signing a consent form, each participant was given instructions explaining the rules of the game. The instructions were similar to those presented here except the information in Table 2 was not provided. As a result, the participant did not know what type of strategies to expect, or when strategy changes would take place. However, the participants were aware that strategies could change during the game.

Participants were given as much time as needed to complete the task and were allowed to consult the instruction sheet during play. All the participants seemed to have no problem understanding the game and no questions were asked while performing the task.

## Models

A total of six cognitive models were written to conduct the study described here. All six models were written using the Herbal high-level language and development environment (Cohen, Ritter, & Haynes, 2005).

The Herbal high-level language is based on the Problem Space Computational Model (PSCM) (Newell, Yost, Laird, Rosenbloom, & Altmann, 1991) and produces productions that can run in both the Soar cognitive architecture (Laird & Congdon, 2005) and Jess (Friedman-Hill, 2003). In this study, the Herbal generated Soar productions were used. However, Jess productions would have also been adequate.

Because of the use of the Herbal high-level language and graphical editor, the creation of the models described here required only an understanding of the PSCM (which provided an infrastructure for model organization) and some visual modeling techniques. This serves as an example of how Herbal can provide modelers without a strong programming background access to the complicated machinery used by architectures that may traditionally be out of their reach. As these models progress towards more sophisticated learning algorithms, the simplified access to Soar and the PSCM will become even more valuable.

All of the models described here are available online as examples at the Herbal website (acs.ist.psu.edu/herbal).

### Batter Models

Five cognitive models were written to represent the strategies used by the batter (Hacker, Aggressive, Random, Chicken, and Alternate). These models are not capable of learning and served only as opponents that exhibit the behavior described in Table 2.

### Pitcher Model

A sixth model was written to play the role of the pitcher. The goal of the pitcher model was to exhibit behavior similar to that demonstrated by the participants. Unlike the batter strategy models, the pitcher model was able to learn using reflection. More specifically, this model operated within two problem spaces: one to deliberate what pitch to throw next, and one to reflect on recent performance and modify future deliberation. The formulation of an explicit reflection phase was simplified by the use of the PSCM and Herbal.

The pitcher model started with an equal probability of throwing a curveball or a fastball. Within the explicit reflection problem space, the pitcher model considers the following features of the environment: the previous number of balls and strikes on the batter, the previous pitch thrown, and the outcome of that pitch. If the outcome is positive (e.g., a strike was called or the batter struck out) the pitcher adjusts a probability so that it is more likely to throw the same pitch the next time it encounters this situation. If, on the other hand, the outcome was negative (a ball or contact by the batter, including contact resulting in an out), and the pitcher had previously experienced a positive outcome in this situation (a strike or a strikeout), the probability of throwing the same pitch in that situation was decreased.

The probability of an action occurring was adjusted by altering working memory so that more or fewer indifferent operators were proposed to throw that pitch type in a given situation. In other words, positive events lead to episodic memory that influences future action. Alternatively, negative events remove episodic memory, reducing this influence. Without prior positive outcomes in a particular situation, no episodic memory elements exist and negative outcomes in that situation are ignored.

An alternative approach to episodic memory would be to use the *numeric-indifferent* preference in Soar (Laird & Congdon, 2005). However, the Herbal high-level language did not support this at the time these models were written.

### Model Parameters

The pitcher model takes two parameters: the learning rate and the unlearning rate. The learning rate specifies how quickly the model will commit to throwing a particular pitch in a particular situation; in other words, how quickly the probability increases given a positive outcome. The unlearning rate specifies how quickly the model will reduce this learned commitment. The best values for these learning rates almost certainly depend on the nature of the particular task.

Considering the relatively simple rules in the baseball task described above, it is expected that participants will be able to learn strategies quickly. In addition, it is hypothesized that participants will at first be reluctant to unlearn until they are sure that a strategy shift has taken place. Given persistent negative feedback on a previously learned response, participants should eventually accelerate their unlearning rate.

Looking at the task environment more closely, further justification of these parameter values can be found in the fact that four of the five batter strategies are deterministic. When a particular pitch works for a batter in a specific situation, it will continue to work until a strategy shift takes place. After a particular pitch stops working for a batter, it can be assumed that a strategy shift has occurred.

As a result, in an effort to match human behavior the pitcher model described here was equipped with a fast learning rate and an initially stubborn, but later accelerating, unlearning rate. Figure 2 depicts the learning and unlearning rates used by the model.

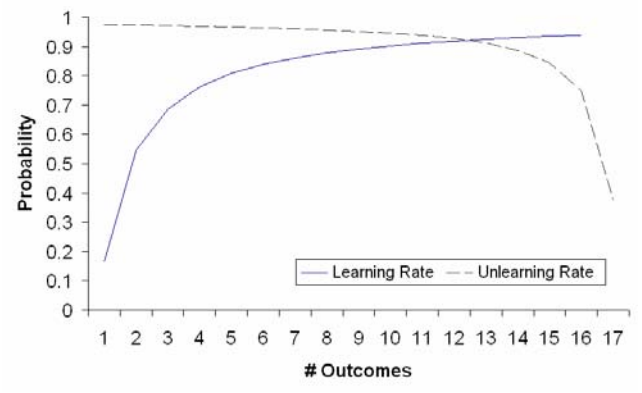


Figure 2: Learning and Unlearning Rates Used by the Model.

## Results

Because a primary goal of this work was to produce a model that not only matches the average pitching efficiency, but also matches the variability in pitching efficiency, the cognitive models created here are not deterministic. This allowed us to consider each run of the model as being equivalent to a participant run. To reduce any sampling error with this theory, the model was run 100 times.

Table 3 shows the results of the participant study and of the model executions. The average pitching efficiency and the standard deviation of the pitching efficiency are listed for all participants and all model runs. Recall that the smaller the pitching efficiency the more efficient the pitcher, and the most efficient strategy has a PE equal to 1.0.

Figure 3 visualizes the data listed in Table 3. Each bar in Figure 3 represents the average pitching efficiency as defined in the Methods section. White bars represent the participant data and shaded bars represent the model data. The error bars in Figure 3 signify one standard deviation from the average pitching efficiency.

Table 3: Pitching Efficiency against Each Batting Strategy for Participants and the Learning Pitching Model.

| | Participants (n = 10) | | Model (n = 100) | |
|---|---|---|---|---|
| **Strategy** | **Mean** | **StdDev** | **Mean** | **StdDev** |
| Hacker [4] | 1.53 | 0.80 | 1.69 | 0.70 |
| Aggressive [7] | 1.81 | 1.62 | 1.13 | 0.20 |
| Random [5] | 5.00 | 6.24 | 5.36 | 4.67 |
| Chicken [4] | 1.03 | 0.08 | 1.25 | 0.33 |
| Alternate [7] | 1.54 | 0.72 | 3.53 | 2.01 |

## Discussion

Analysis of Figure 3 reveals that the model's behavior matched both the participant's average performance, and variability in performance, for three of the five presented strategies. However, for two of the strategies the model did not satisfactorily reflect the participant's performance.

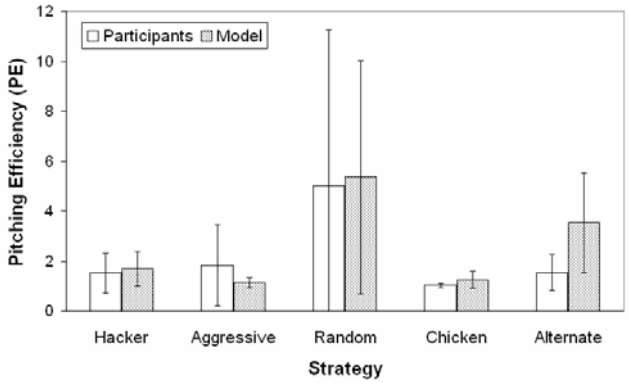


Figure 3: Comparison of Learning Pitching Model and Participants for the Batting Strategies. SDs are shown as error bars.

### Hacker and Chicken Strategies

The model's performance matched very well for both the Hacker and Chicken strategies. Given the simplicity of the learning strategy used, this is an interesting result. Both the participants and the model were able to retire the requisite number of consecutive batters quickly and without much variability. Interestingly, the Hacker strategy proved to be more difficult for both the participants and the model. This may be because the very aggressive strategy used by the Hacker makes it more likely for the batter to get a hit when the pitcher made a mistake. On the other hand, the reserved approach used by the Chicken strategy only punishes mistakes with a single ball as opposed to a hit. In this baseball task, an aggressive batter strategy is more dangerous to the pitcher than a timid one.

### Random Strategy

As expected, the variation of the pitching efficiency against the Random strategy was quite large for both the participants and the model. Both the participant and the model could not consistently figure out the random strategy, because, well, it was random. The difference between the pitching efficiency for the model, and that of the participants, might be related to the number of participants run. Due to the random nature of this strategy, additional participants might cause these averages to match more closely.

### Aggressive and Alternate Strategies

Unexpectedly, the model did not do as good of a job matching the Aggressive and Alternate strategies. The order in which these strategies are presented may play an important role here. One possible explanation for these problems is that the unlearning rate used by the model is not fast enough. While good enough to match the transitions between some strategies, the unlearning rate may need to be faster in other cases. To understand this theory, the transitions from the Hacker strategy to the Aggressive strategy, and from the Chicken strategy to the Alternate strategy, need to be examined more closely.

**Transition From Hacker to Aggressive** Because the Hacker strategy always swings, the pitcher must learn to throw a series of curveballs to get a batter out consistently. In addition, the inability to quickly unlearn the Hacker strategy is not immediately detrimental when an Aggressive batter follows the Hacker strategy. For example, if the pitcher continues to throw a series of curveballs to an Aggressive batter, the batter will not get on base until after the sixth curveball is thrown. This gives the pitcher several pitches, and therefore a lot of time to unlearn the strategy.

On the other hand, if the participant or model quickly unlearns the Hacker strategy, it will lead to throwing an early fastball which will result in a 50% chance of the batter getting a base hit. In other words, in this particular case quickly unlearning the previous strategy is not beneficial. This might explain why the model performed better against the Aggressive strategy; the model simply does not unlearn as quickly as the participants did, and this proved to be more efficient in this particular ordering of strategies.

**Transition From Chicken to Alternate** The opposite can be said about the transition from the Chicken strategy to the Alternate strategy. A series of consecutive fastballs will get a batter out using the Chicken strategy because this strategy never swings. However, if this knowledge goes unlearned, the same series of fastballs thrown to an Alternate batter will result in frequent hits because the Alternate batter swings immediately after a fastball is thrown. In this particular case, failure to quickly unlearn the Chicken strategy results in poor performance and might explain why the model did not perform as well as the participants in this case. Once again, it appears as if the model did not unlearn the learned strategy quickly enough in this particular ordering of strategies.

Unfortunately, our reflective learning strategy is fundamentally limited in how quickly it can unlearn. This limit may be a major reason for the model's inability to unlearn the Chicken strategy quickly enough. Recall that the learning algorithm used here cannot unlearn unless it has already encountered positive feedback. This causes a problem if the model's initial encounter with a strategy involves a series of negative outcomes, which is precisely the case when transitioning from Chicken to Alternate. Augmenting the algorithm to use Soar's *numeric-indifferent* preference might eliminate this limitation and possibly improve the model's fit.

**Additional Explanations** Factors other than unlearning rate may have also had an effect on the model's inability to match the participant's behavior. For example, if the pitcher follows the simple pattern of throwing a fastball, followed by curve, followed by fastball, they will always get the Alternate batter out. While speculative, it is possible that participants were quick to recognize this alternating pattern while the model did not treat alternating patterns any differently from other patterns.

## Conclusions

The paper describes a study to measure the learning of participants performing a competitive task. This task is based on the game of baseball and consists of a participant pitching to a series of batters that use a set of different strategies. Because this task is inspired by the example introduced in *A Gentle Introduction to Soar,* many Soar programmers may already be familiar with its attributes. One outcome of this work is that, over time, the continued use of baseball as a running example might help beginners learn modeling.

In addition, this paper introduced a Soar model written using the Herbal high-level language. This model used a reflective learning process to learn and unlearn various strategies. Another outcome of this work is a downloadable example of how Soar models that learn can be written without directly writing Soar productions. Easily obtainable examples like this will hopefully make Soar models available to a wider audience.

The model's behavior was compared to participants' performance and was shown to match both the participants' average performance and variability in performance against many of the presented batting strategies. This result demonstrates that cognitive models that compete in adversarial environments using introspective learning need not be complicated and can be written quickly and easily using Herbal.

Finally, for the strategies that the model did not satisfactorily master, insight into the limitations of the algorithm used, and how people possibly perform this task was gained. The relationship of the sequences of strategies and how learning is transferred was explored. These results motivate future work that will lead to improvements in the learning algorithm, and in the Herbal high-level language.

## Future Work

The results reported here provide some insights to guide future work. To start, the limitations of the learning algorithm discovered here can be addressed by exploring more sophisticated learning mechanisms (e.g. the meta-learning routines described in (Sun, 2001)).

Further work can also be done to alter the reflection strategy so that certain patterns are easier to learn than others. Patterns that people recognize quickly (such as alternating patterns) might create more intense episodic memories in the model. This change would test the theory that the participants performed well in cases where the solution consisted of a simple and quickly recognizable alternating pattern.

Additional improvements to the model could also be made by enhancing the reflective process so that positive experiences are no longer required in order to benefit from negative experiences. In the absence of positive learned events, negative reflection should still lead to a decrease in the probability of repeating the action. One solution could involve equally increasing the probability of all other possible actions when an event results in a negative outcome. This would be easier to accomplish if the Soar *numeric-indifferent* preference was used to control operator probabilities, and this capability is currently being added to the Herbal high-level language.

There is also scope to explore other parts and versions of the baseball task. For example, the environment and models can be expanded to include other batting strategies, other batter sequences, batting tournaments, and learning batters. In addition, the model created here could be transformed into an Herbal library that can be reused in future models.

Finally, because the Herbal development environment automatically creates both Soar and Jess models, the opportunity exists for comparisons of a single Herbal high-level model running in two very different architectures.

## Acknowledgments

The development of this software was supported by ONR (contract N00014-06-1-0164). Comments from Mary Beth Rosson and Richard Carlson have influenced this work. Recognition is also given to the undergraduate students that agreed to participate in this study.

## References

Bass, E. J., Baxter, G. D., & Ritter, F. E. (1995). Creating models to control simulations: A generic approach. *AI and Simulation Behaviour Quarterly, 93*, 18-25.

Cohen, M. A., Ritter, F. E., & Haynes, S. R. (2005). Herbal: A high-level language and development environment for developing cognitive models in Soar. *In Proceedings of 14th Behavior Representation in Modeling and Simulation,* 133-140. University City, CA.

Cox, M. T., & Ram, A. (1999). Introspective multistrategy learning: On the construction of learning strategies. *Artificial Intelligence, 112*, 1-55.

Friedman-Hill, E. (2003). *Jess in action: Rule-based systems in Java*. Greenwich, CT: Manning Publications Company.

Jones, R. M., Laird, J. E., Nielson, P. E., Coulter, K. J., Kenny, P., & Koss, F. V. (1999). Automated Intelligent Pilots for Combat Flight Simulation. *AI Magazine, 20,* 27-41.

Laird, J. E. (2001a). It knows what you're going to do: Adding anticipation to a Quakebot. *In Proceedings of Fifth International Conference on Autonomous Agents,* 385-392. New York, NY: ACM Press.

Laird, J. E. (2001b). Using a computer game to develop advanced AI. *IEEE Computer, 34*(7), 70-75.

Laird, J. E., & Congdon, C. B. (2005). *The Soar User's Manual Version 8.6*: The Soar Group: University of Michigan.

Lehman, J. F., Laird, J. E., & Rosenbloom, P. S. (1996). A gentle introduction to Soar: An architecture for human cognition. In D. Scarborough & S. Sternberg (Eds.), *An invitation to cognitive science* (Vol. 4). New York: MIT Press.

Newell, A., Yost, G. R., Laird, J. E., Rosenbloom, P., & Altmann, E. (1991). Formulating the problem space computational model. In R. F. Rashid (Ed.), *Carnegie Mellon Computer Science: A 25-Year commemorative* (pp. 255-293). Reading, MA: ACM-Press (Addison-Wesley).

Ritter, F. E., Nerb, J., O'Shea, T., & Lehtinen, E. (Eds.). (2007). *In order to learn: How the sequence of topics influence learning*. New York: Oxford University Press.

Ritter, F. E., Shadbolt, N. R., Elliman, D., Young, R., Gobet, F., & Baxter, G. D. (2002). *Techniques for modeling human performance in synthetic environments: A supplementary review*. Wright Patterson Air Force Base, OH: Human Systems Information Analysis Center.

Ritter, F. E., & Wallach, D. P. (1998). Models of two-person games in ACT-R and Soar. *In Proceedings of Second European Conference on Cognitive Modeling,* 202-203. Nottingham: Nottingham University Press.

Russell, S., & Norvig, P. (2003). *Artificial Intelligence: A Modern Approach* (2nd ed.). Upper Saddle River, NJ: Prentice Hall.

*SML Quick Start Guide*. (2005): ThreePenny Software LLC.

Sun, R. (2001). Meta-learning processes in multi-agent systems. *In Proceedings of Intelligent Agent Technology,* 210-219. Maebashi, Japan: World Scientific, Singapore.

Sun, R. (Ed.). (2006). *Cognition and multi-agent interaction*. Cambridge University Press: New York.