ADAPT: A Cognitive Architecture for Robotics

D. Paul Benjamin (benjamin@pace.edu)
Computer Science Department, Pace University
1 Pace Plaza, New York, NY 10038 USA

Damian Lyons (dlyons@fordham.edu)
Computer Science Department, Fordham University
340 JMH Fordham University, 441 E. Fordham Rd., Bronx, NY 10458

Deryle Lonsdale (lonz@byu.edu)
Linguistics Department, Brigham Young University
Brigham Young University, Provo, UT 84602 by January 15, 2004

Abstract

ADAPT (Adaptive Dynamics and Active Perception for Thought) is a cognitive architecture specifically designed for robotics. The ADAPT architecture is in the initial state of development. ADAPT manipulates a hierarchy of perceptual and planning schemas that include explicit temporal information and that can be executed in parallel. Perception is active, which means that all perceptual processing is goal-directed and context-sensitive, even down to the raw sensory data. This paper describes the components of ADAPT and how it differs from a number of existing cognitive architectures.

Introduction

A number of cognitive models have been developed, including EPIC (Kieras, Wood & Meyer, 1997), Soar (Laird, Newell & Rosenbloom, 1987), and ACT-R (Anderson, 1996). These models were developed specifically to model human performance on a range of tasks. The models were evaluated based on the degree to which they fit human performance data on these tasks, especially timing data. These tasks typically were solving puzzles of various types and performing simple motor tasks, or learning basic cognitive tasks such as word recognition or sentence disambiguation. The model that incorporates a detailed model of perception (EPIC) faithfully reproduces the restrictions and timings of human perception (particularly eyes and ears). Other architectures such as Soar and ACT-R have borrowed this model from EPIC.

The motivation for the development of the ADAPT cognitive architecture comes instead from robotics: researchers want their robots to exhibit sophisticated behaviors including use of natural language, speech recognition, visual understanding, problem solving and learning, in complex analog environments and in real time. A growing number of robotics researchers have realized that programming robots one task at a time is not likely to lead to a robot with such general capabilities, so interest has turned to cognitive robotic architectures as a natural way to try to achieve this goal. ADAPT is an architecture under development as part of a collaborative research effort on robot cognition combining Pace University's Robotics Lab – csis.pace.edu/robotlab, Fordham University's Robot Lab – www.cis.fordham.edu/~lyons/rcvlab and the linguistics department of Brigham Young University – linguistics.byu.edu/nlsoar.

Robotics researchers are faced with a hurdle when attempting to use cognitive architectures to control robots: the architectures do not easily support certain paradigms of perception and control that are mainstream in robotics. This paper focuses on two such paradigms: adaptive dynamics and active perception.

The design of ADAPT is distinguished by two assumptions. The first is an assumption about the nature of perception that differs from that of existing cognitive architectures. Perception is modeled as an *active* process, rather than as a passive one. This means that perception is goal-directed and context-sensitive at every stage, including the initial processing of input sensory data. Active perception processes even low-level visual and auditory data in a goal-directed manner. For example, if a robot is crossing the street, it will not scan the entire visual field, but rather turn its head in the direction of oncoming traffic. It will not process all the parts of the resulting visual frame equally, but will filter the data for large blobs that are moving towards the robot, ignoring (or processing in a very coarse way) the rest of the visual field. Such a robot will perceive cars very differently depending on the goal and the situation.

Active perception, and in particular active vision (Blake & Yuille, 1992), is a major research area in robotics. Existing cognitive architectures can be altered to perceive in this way (Byrne, 2001) but this is the exception rather than the rule. EPIC in particular does not perceive this way, and this is a major limitation in its application to robotics.

The second assumption is that true parallelism is necessary and that a robot must be able to reason about a hierarchy of concurrent real-time actions. A robot typically possesses a large number of moving components, e.g. gripper joints, wheels or legs, pan-tilt camera units, camera lenses that zoom, and these components may all be in action simultaneously, in addition to actions occurring simultaneously in the environment.

Many existing robot control paradigms consist of a

hierarchy of behaviors, e.g. (Brooks, 1991) and (Lyons and Hendriks, 1995). Existing cognitive architectures do not facilitate implementing such hierarchies.

ACT-R permits parallel perception and parallel operation of motors, but has typically not been used to reason about concurrent actions, and certainly not in a hierarchy. Soar permits parallelism of a sort. Typically, actions in Soar are modeled as operators, and Soar can execute only one operator at a time per problem space; multiple spaces can be active at a time, but the goals for these spaces must be arranged in a stack. It is difficult, although not impossible, to implement a hierarchy of behaviors in Soar. We need a more flexible arrangement of goals that permits multiple abstract behaviors that can share implementations. ADAPT controls perception and action using schemas that are represented declaratively in working memory. Multiple schemas can be present and active simultaneously, and can be connected into networks, thereby permitting the robot to reason about a hierarchy of concurrent actions.

A deeper reason for the development of this architecture is that it allows us to examine more deeply the interrelationship between perception, problem solving and the use of natural language. In the four architectures mentioned above, these three aspects of cognition are separate. In particular, perception is a peripheral activity that is clearly separated from problem solving. We need to permit researchers to explore issues such as how perception is related to representation change in problem solving, and how linguistic structures may affect problem solving. ADAPT is an architecture intended to explore the integration of perception, problem solving and natural language at a deeper structural level.

The next section of this paper describes in more detail these two limitations of existing architectures for robotics, and the subsequent sections describe ADAPT. In many respects, ADAPT is similar to ACT-R and in other respects it resembles Soar. After discussing the motivations for our architecture, we describe how ADAPT generalizes the search control mechanisms of Soar and ACT-R.

Previous Work

The development and implementation of unified theories of cognition (Newell, 1990) has become an important part of cognitive science in the last twenty years. These architectures strive to explain, implement and measure a range of human cognitive activities. The three preeminent architectures are EPIC, Soar and ACT-R. Each of these has achieved a degree of success and is used in one or more applications. Although a detailed examination of these systems is outside the scope of this paper, we will briefly examine the perceptual systems and planning structures employed by them.

Each of these architectures is based on a set of productions that are matched against a working memory. One or more of the productions that match are fired, with the results affecting working memory and possibly causing external effects. The systems vary according to how the productions are chosen, and whether parallel firing of productions is permitted. ACT-R chooses a single production to fire. EPIC permits only one action production to fire. Soar selects only one operator at a time in each problem space to perform actions.

The EPIC cognitive architecture was specifically developed to model human perception, and thus has a well developed perceptual model. This model has been largely imported into ACT-R, and also serves as the perceptual basis for EPIC-Soar (Chong, 1998), so an examination of EPIC covers the basics of perceptual modeling in all three architectures, although Soar itself has no perceptual model and is unconstrained.

EPIC's perceptual model includes simulations of human eyes and ears, as well as basic motor functions and some tactile sensing. These models include details such as the sizes of various retinal zones. There is no direct provision for non-human modalities, e.g., laser range finding. All perceptual mechanisms fire in parallel, and their output is placed in working memory. This output is only the final output of visual and auditory pattern recognizers. There is no access to sensory data or to the pattern recognition algorithms, so the pattern recognition is fixed and goal-independent. This prevents use of active perception.

The three architectures vary considerably in their approach to goals and problem solving. EPIC is the simplest, as it has no mechanism for choosing between actions. The human programmer must model the actions of the system so that no conflicts are possible. Parallel execution of actions is possible, but only if the actions that use distinct peripheral processors. There is no planning and goals do not determine the system's actions, so that by itself EPIC is not adequate for robotics.

Soar maintains an explicit goal stack. The actions that are possible at each step (the "operators") are attached to the state corresponding to each goal. Soar subgoals (in a sense, calling itself recursively by creating a new goal and state) to select an operator when more than one is possible. Typical Soar programming models the task actions of the system as operators, which means that only one action is possible per goal at a time. This severely restricts Soar's usefulness in robotics, as the system can have difficulty reasoning about a hierarchy of concurrent actions and concurrent goals that is not organized as a stack (Laird & Rosenbloom, 1990).

ACT-R forbids parallelism, and uses a Bayesian mechanism to select one preferred action. It maintains a goal stack that is part of this determination, so that only one goal is active at a time. ACT-R has a long-term declarative memory (Soar does not) and declarative "chunks" of information are retrieved from long-term declarative memory according to the strength of their association with the current goal and contents of working memory. One production is chosen to fire based on its utility. As with Soar and EPIC, ACT-R does not permit two independent actions to be executed in parallel, which can give it difficulty representing plans.

Our Approach to Perception and Planning

ADAPT resembles the above architectures in many ways. It is a production-based architecture with a working memory, and matches productions against working memory during each cycle in the same manner as these architectures. All the matching productions fire, as in Soar, and place their results in working memory. ADAPT possesses a long-term declarative memory, as ACT-R does, in which it stores sensory-motor schemas that control its perception and action. All perceptual processors fire in parallel, as in EPIC, but place their low-level sensory data into working memory, where it is processed by the cognitive mechanism. Thus, ADAPT strongly resembles EPIC-Soar, with the additional property that general schemas are stored permanently in working memory.

Robots are not limited to human sensors or effectors, e.g. robots can have lasers and wheels and non-human grippers. So ADAPT generalizes some of the structures found in other cognitive architectures, as well as relaxing the adherence to human timing data for the performance of non-human sensors or effectors.

EPIC and ACT-R include a processing bottleneck, which is that the architecture can process only one goal at a time and execute only one action at a time. (EPIC has this bottleneck, but the programmer must enforce it.) In ADAPT, we could have dropped this bottleneck and permitted the architecture to have multiple goals active at the same time (so the goals would not be organized on a stack), and permitted the architecture to execute multiple actions simultaneously. After all, our stated goal is to do just that. However, we have included this bottleneck in ADAPT, too, and it is necessary for us to explain why.

We draw a distinction between the goals that are task goals, e.g. "find the blue block", and those that are goals of the architecture, e.g. "start the schema that scans the environment for a segment of a given color". Similarly, we distinguish between task actions, e.g. "pick up the block", and architectural actions, e.g. "start the gripper-closing schema". Our goal is to reason about concurrent goals and actions *in the task*. As we have pointed out, the task environment is complex; there are many things to be perceived, and complex actions to be coordinated. But architectural goals and actions cannot be allowed to multiply in the same fashion as task goals and actions, because the complexity of the architecture would become too great; the architecture needs to be focused on one goal at a time and to execute one action at a time. This not only leads to simplicity of implementation (avoiding the need for a full-blown implementation of something like CSP (Schneider, 1999)), but also keeps the architecture from spending its time multiplying goals rather than solving them.

We accomplish this dichotomy by partitioning the actions and goals into an architectural part consisting of architectural goals and actions and a task part consisting of task-specific goals and actions. We restrict the architectural part to one active goal (it has a goal stack) and one architectural action at a time. The architectural part is procedurally represented, i.e. the system can execute the actions but cannot examine their internal representation. We represent the task goals and actions declaratively in working memory as well as procedurally. There can be as many active task goals and actions in working memory as the system wants. We call these goals and actions "schemas".

Schemas

The term "schema" has a long history in artificial intelligence, cognitive modeling and brain science, which we do not have space to recapitulate in full here. Minsky's frames are a type of schema, as are Schank's scripts. Arbib (1992) has developed a schema theory that essentially describes our use of the term. His schema theory is especially useful because it has been precisely specified with a formal semantics (Steenstrup, Arbib & Manes, 1983) and implemented in a language for real-time concurrent perception and action (Lyons & Arbib, 1989). We use this language, called RS (for Robot Schemas), to give our cognitive model the concurrent, real-time reasoning capability that it needs.

One very important aspect of schemas is that they combine procedural and declarative knowledge, i.e. all knowledge in a schema can be accessed declaratively for purposes such as communication and generalization, and accessed procedurally to solve problems. This distinguishes schemas from Soar's operators or ACT-R's declarative chunks. This combination of procedural and declarative representations is of central importance, as this permits behavioral knowledge to be both executed in procedural form as well as communicated by natural language in declarative form. Additionally, representing schemas in declarative form permits ADAPT to reason about and transform the hierarchy of schemas. For example, ADAPT can apply inductive inference operators to the declarative forms of schemas to generalize them.

Schemas are general patterns of perception and action, and are stored permanently in working memory. Each schema has a sensory part and a motor part, representing the goal of executing a particular motion depending on certain perceptual conditions. The motor parts of the schemas are the task actions. Schemas also contain explicit temporal information, e.g. about intervals of time during which an action must take place. General schemas are instantiated to create schema instances that are connected to each other to form a network. A schema can be abstract, and either the sensory or motor part can be absent (or trivially true). The part of working memory that contains the library of schemas that can be instantiated and activated is similar in a sense to the declarative memory where ACT-R stores its chunks.

Schemas control the actual motors as a side effect of placing control information into working memory, which causes the robot to move. Since multiple schemas can be active simultaneously, parallel schema executions are possible. It is the responsibility of the system (while it is planning) to ensure that simultaneous executions can be

executed together. As only one architectural action is executable at a time, the architecture cannot start two schemas simultaneously or instantiate the variables of two schemas simultaneously. Specifically, it cannot initiate the processing of two different modes of sensory input at the exact same time. This respects the cognitive data available on dual tasking (Gopher & Donchin, 1986), so that compatibility is retained to some extent with human cognitive data. But this does not result from a specific decision to retain this compatibility, but from the decision to retain the processing bottleneck to keep the system focused.

ADAPT plans by transforming a hierarchy of schemas (there is a root schema "interact with the environment" to tie all the schemas together.) At each step, ADAPT can perform one of the following steps:

- refine a schema into subschemas,
- instantiate variables in a schema (this includes connecting two or more schemas by binding their variables together),
- start execution of a schema,
- suspend execution of a schema, or
- terminate and remove a schema.

Thus, ADAPT operators work at the executive level rather than at the task level, continually modifying the schema hierarchy. Task-level actions are executed by the motor parts of the schemas.

ADAPT is flexible enough to implement general concurrent real-time robot programs. The behavioral approach we favor is not a traditional planning approach but rather one of "adaptive dynamics" in which the system tries to use one or more simple behaviors to generate a complex behavior (Benjamin, 2000; Lyons & Hendriks, 1995). For example, if the robot is trying to pick up a ball that is rolling on the floor, it activates three schemas: the grasping schema, the move-arm schema and the tracking schema. The grasping and move-arm schemas are connected to the tracking schema. As the rolling ball is tracked by the tracking schema, information is fed independently to the two motor schemas to control the rate of grasping and movement, so that the ball will be caught. This is very different from and much simpler than planning the effects of interleavings of small arm and finger movements.

This approach relies very heavily on the representation used by the system; we give ADAPT the ability to reformulate its knowledge to try to find simple schemas that generate complex behaviors (Benjamin, 1992; Benjamin, 1994). One of the central research goals of this project is to investigate mechanisms for solving problems by reformulation, as explained by Duncker (1945).

Search Control in ADAPT

ADAPT possesses a very general mechanism for selecting one action to perform. This mechanism is essentially the same as Soar's subgoaling mechanism, because the emphasis in ADAPT is on problem solving by search, as it is in Soar. The philosophy behind this mechanism is that the system should be capable of bringing all its knowledge to bear on the central issue of selecting its actions. ACT-R does not permit this, instead using a Bayesian mechanism to select actions based on their anticipated gain.

However, much robotics research uses such nonsymbolic mechanisms. In particular, there is a very large body of robotics research using neural networks, so a general robotic cognitive architecture should be able to model nonsymbolic reasoning and learning mechanisms, which Soar does not. Although the mechanism ADAPT uses to select actions is Soar-style subgoaling, the presence of the declarative schema memory permits ADAPT to implement ACT-R nonsymbolic mechanisms, too. This is accomplished by storing the information that ACT-R uses to compute utilities with the declarative schemas, so that each schema has stored with it the estimate of the probability that it will lead to eventual achievement of the current goal and the estimate of the cost to achieve the goal. Both probability and cost are measured in time, as in ACT-R. The utility computation is performed explicitly with ADAPT actions, rather than in the mechanism as ACT-R does.

When selecting among multiple schemas, ADAPT reaches an "impasse" just as Soar does and subgoals to choose one schema. It can choose either to use the Bayesian estimate or to search. This decision can be made based on factors such as the time available. This generalization of the selection mechanisms of ACT-R and Soar permits ADAPT to model flexible selection strategies that combine symbolic and nonsymbolic processing.

For example, in the example given above of the rolling ball, the robot might need to decide whether to take a step closer to the path of the rolling ball to make it easier to pick up. If this decision must be made immediately, the robot will use its estimated gain for such movements in general. If there is time to deliberate, the robot can reason about the length of its arm and how far it is from the path of the ball to decide if it will have a good chance to succeed.

Learning in ADAPT

There are two different methods of learning in ADAPT: procedural learning of search control and inductive inference of schemas.

ADAPT utilizes a method of learning search control that is borrowed from Soar. ADAPT generates procedural "chunks" when goals are satisfied in exactly the same way that Soar does. These chunks are productions whose left-hand sides contain all the working memory elements that were referenced in making the search-control decision, and whose right-hand side is the decision. It is well understood how Soar speeds up its search control by learning search control chunks (Rosenbloom, Laird & Newell, 1993).

A search-control chunk that ADAPT learns may use the Bayesian estimate to make the choice of action, in which case the chunk performs in one step the same choice that ACT-R would make. Or the chunk may compile the results of a search of alternatives, in which case the chunk performs

just as a Soar chunk does.

ADAPT can learn a chunk that combines these two methods to select a schema based on a combination of gain and search.

The declarative representation of schemas permits ADAPT to transform schemas in many ways. In particular, ADAPT can perform inductive inference on schemas, e.g. by replacing a constant with a variable or by enlarging an interval of permitted numeric values. These changes are performed by operators that examine the execution history and hypothesize more general schemas that are added to the declarative memory. As these new schemas are tried in future situations, their successes and failures are recorded in the schemas.

An Implementation of ADAPT

An initial implementation of ADAPT has been completed. Testing has just begun using a Pioneer P2 robot that is equipped with stereo color vision, microphone and speakers, sonars and touch sensors.

The implementation is within the Soar system, because of the similarity between ADAPT and Soar. A declarative memory has been added to Soar for the general schemas, together with a set of operators that instantiate a general schema and transform the instance into a set of productions. Additional operators have been added to start, pause and stop schemas. Schemas are executed by a runtime system that implements the RS schema system (Lyons & Arbib, 1989) in the Colbert language, which is a behavior-based language created by Kurt Konolige and provided with Pioneer robots.

Currently, ADAPT has access only to sonar and bump-sensor readings. This permits simple navigation and pushing behaviors, but nothing more. Our plans are to complete the integration of language and vision capabilities within two months. The existing version of ADAPT has a cycle time of 50ms and is very successful at guiding the robot at basic navigation tasks such as moving from one room to another and avoiding obstacles.

The language component will be provided by NL-Soar (Lonsdale, 1997; Lonsdale, 2000; Lonsdale, 2001), which is currently being ported to Soar8 from Soar7 as part of our collaboration with Brigham Young University. It will be available in May and will be integrated into the system.

The vision component consists of two pieces: a bottom-up component that is always on and is goal-independent, and a top-down active component. Both components exist but the full integration within ADAPT is not complete. We anticipate a full integration by July.

Testing and Evaluating ADAPT

We have selected an important and flexible class of mobile robot applications as our example domain: the "shepherding" class. In this application, one or more mobile robots have the objective of moving one or more objects so that they are grouped according to a given constraint. An example from this domain is for a single mobile platform to push an object from its start position over intermediate terrain of undefined characteristics into an enclosure. Another example is for it to push a set of objects of various shapes and size all into the enclosure. A more complex example is to group objects so that only objects of a particular color are in the enclosure.

This class of tasks is attractive for two reasons. The first is that it includes the full range of problems for the robot to solve, from abstract task planning to real-time scheduling of motions, and including perception, navigation and grasping of objects. In addition, the robot must learn how to push one or more objects properly. This range of demands is ideal for our purposes, because it creates a situation in which complex hierarchies of features and constraints arise. Also, the tasks can be made dynamic by including objects that can move by themselves, e.g. balls or other robots.

The second reason is that we can embed lots of other problems in it, especially those that have been examined by cognitive psychology. For example, we can create an isomorph of the Towers of Hanoi task by having three narrow enclosures and adding the constraint that no object can be in front of a shorter object (so that all objects are always visible by the observer). (see Figure 1) The three enclosures act as the three pegs in the Towers of Hanoi, and the constraint is isomorphic to the constraint that no disk can be on a smaller disk in the Towers of Hanoi. The robot (gray) must move the three boxes from the leftmost column to the rightmost. It can push or lift one box at a time. The front area must be kept clear so the robot can move; there can be no boxes left in this area.

This creates a situation in which the robot can be presented with a Towers of Hanoi problem in a real setting with perceptual and navigational difficulties, rather than just as an abstract task. This permits us to evaluate the robot's problem-solving and learning capabilities in a way that permits comparison with human data.

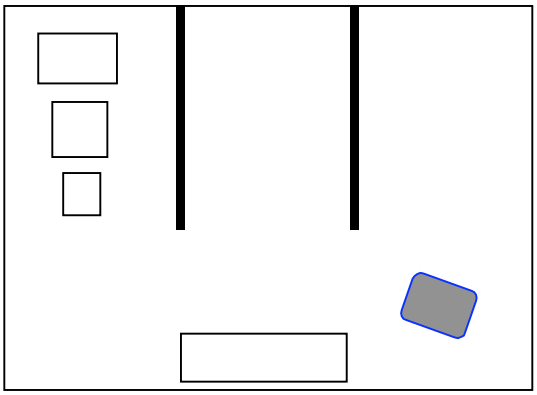

Figure 1

Similarly, we can embed bin-packing problems by making them enclosure-packing problems. Also, we can

embed sorting problems, and embed block-stacking problems. In this way, many existing puzzles and tasks can be embedded in a realistic setting and posed to the robot.

Summary

The fields of cognitive psychology and robotics have much to offer each other. The development of robot cognitive architectures is an attempt to apply the results of cognitive modeling to the difficult problems faced by robotics research. ADAPT is a cognitive architecture specifically designed to permit robotics researchers to utilize well-known robotics research in areas such as active perception and adaptive dynamics within a cognitive framework. This architecture is still in its infancy, and in particular has not yet been integrated with vision or language. The goals of this research are to expand the capabilities of robots and simultaneously to expand and generalize the capabilities of existing cognitive models.

Acknowledgements

We would like to thank Frank Ritter for many helpful comments and pointers to relevant work.

References

Anderson, J. R. (1996). ACT: A simple theory of complex cognition. *American Psychologist* , 51, 355-365.

Arbib, M. A. (1992). Schema Theory, in S. C. Shapiro (Ed.), *Encyclopedia of Artificial Intelligence (2nd edition)*, Wiley-Interscience.

Benjamin, D. Paul, (1992). Reformulating Path Planning Problems by Task-preserving Abstraction, *Journal of Robotics and Autonomous Systems*, 9, pp.1-9.

Benjamin, D. Paul, (1994). Formulating Patterns in Problem Solving, *Annals of Mathematics and AI*, 10, pp.1-23.

Benjamin, D. Paul, (2000). On the Emergence of Intelligent Global Behaviors from Simple Local Actions, *Journal of Systems Science*, special issue: Emergent Properties of Complex Systems, Vol. 31, No. 7, pp.861-872.

Blake, A., and Yuille, A., (Eds,) (1992). *Active Vision* , MIT Press, Cambridge, MA.

Brooks, R.A., (1991). How to build complete creatures rather than isolated cognitive simulators, in K. VanLehn (ed.), *Architectures for Intelligence*, pp. 225-239, Lawrence Erlbaum Assosiates, Hillsdale, NJ.

Chong, R.S. (1998). Modeling dual-task performance improvement with EPIC-Soar. *Proceedings of the Twentieth Annual Conference of the Cognitive Science Society,* Hillsdale, NJ, Lawrence Erlbaum.

Duncker, Karl, (1945, 1972). On Problem Solving, in Dashiell, John, (Ed.) *Psychological Monographs*, Greenwood Press, Westport, CT.

Gopher, D., & Donchin, E., (1986). Workload: An Examination of the Concept, in K.R.Boff, L. Kaufman, & J.P.Thomas (Eds.), *Handbook of Perception and Human Performance, Vol. II: Cognitive Processes and Human Performance*, pp. 41.1-41.49, New York, Wiley.

Kieras, D.E., Wood, S.D., and Meyer, D.E. (1997). Predictive engineering models based on the EPIC architecture for a multimodal high-performance human-computer interaction task, *ACM Transactions on Computer-Human Interaction* 4, 230-275.

Laird, J.E., Newell, A. and Rosenbloom, P.S., (1987). Soar: An Architecture for General Intelligence, *Artificial Intelligence* 33, pp.1-64.

Laird, J. E., & Rosenbloom, P. S. (1990). Integrating, execution, planning, and learning in Soar for external environments. *Proceedings of the 8th National Conference on Artificial Intelligence* (pp. 1022-1029). Hynes Convention Centre: MIT Press.

Lonsdale, Deryle, (1997). Modeling Cognition in SI: Methodological Issues, *International journal of research and practice in interpreting*, Vol. 2, no. 1/2, pages 91-117; John Benjamins Publishing Company, Amsterdam, Netherlands.

Lonsdale, Deryle, (2000). Leveraging Analysis Operators in Incremental Generation, *Analysis for Generation: Proceedings of a Workshop at the First International Natural Language Generation Conference*, pp. 9-13; Association for Computational Linguistics; New Brunswick, NJ.

Lonsdale, Deryle, (2001). An Operator-based Integration of Comprehension and Production, *LACUS Forum XXVII*, pages 123–132, Linguistic Association of Canada and the United States.

Lyons, D.M. and Arbib, M.A., (1989). A Formal Model of Computation for Sensory-based Robotics, *IEEE Transactions on Robotics and Automation* 5(3), June.

Lyons, D.M. and Hendriks, A., (1995). Exploiting Patterns of Interaction to Select Reactions, *Artificial Intelligence* 73, Special Issue on Computational Theories of Interaction, pp.117-148.Newell, Allen, (1990). *Unified Theories of Cognition*, Harvard University Press, Cambridge, Massachusetts.

P. S. Rosenbloom, J. E. Laird, and A. Newell, (1993). *The SOAR Papers*. MIT Press.

S. Schneider, (1999). *Concurrent and Real-time Systems: The CSP Approach*. Wiley.

Martha Steenstrup, Michael A. Arbib, Ernest G. Manes, (1983). Port Automata and the Algebra of Concurrent Processes. *JCSS* 27(1): 29-50.