An Interdisciplinary Theory of Autonomous Action

Richard Levinson
Recom Technologies
NASA Ames Research Center
Mail Stop: 269-2, Moffett Field, CA 94035

e-mail: levinson@ptolemy.arc.nasa.gov

Introduction

This paper presents a theory of autonomous action that combines the perspectives of three disciplines: neuropsychology, AI planning, and control system technology. Each discipline brings complementary contributions to the common goal of understanding the nature of autonomous action, and the combined theory requires extensions to each of the component theories. Our overall goal is to: *Extend AI planning methods to perform the functions of the neuropsychological models within the context of real-time, closed-loop control applications.*

The model presented in this paper provides a mapping between the theories that supports the further interdisciplinary exchange of ideas. Related interdisciplinary efforts have been discussed by Levinson[13, 14, 15] and by Spector, Grafman, and Hendler [21, 8, 9].

We begin by considering humans as the "gold standard" for autonomous systems. Human autonomy requires the integration of *default* and *deliberate* behavior. Default behavior is effective in *routine* situations, but it often must be deliberately modified for use in *novel* situations. The default programs are well-learned, automatic reactions that are driven by the presence of strong sensory data. In contrast, the deliberate programs are unlearned, novel responses that are driven by goals in the absence of strong sensory stimuli. The flexibility to deliberately modify default behavior allows humans to function in novel conditions. People with frontal lobe damage often have difficulty modifying default behavior in novel conditions and therefore have difficulty living socially-independent lives. They often respond mechanically to immediate sensory stimuli without regard to the effects of their actions. These patients can score well on IQ tests, but have difficulties planning and executing daily activities for shopping, cooking, washing and business.

We propose that runtime *deliberation* is required for autonomous control systems to perform successfully in novel operating conditions. We define the term *deliberation* to mean a time-consuming decision making process that considers the future. The word deliberation comes from the term "de libra", which means from weight or measurement. Thus, a deliberate choice is a measured selection, where the choices are measured against a set of goals. Without deliberation, modern control systems are also data-driven and reactive, much like frontal lobe patients.

At NASA, we are developing autonomous instruments and spacecraft that require the flexibility of humans to operate in unpredictable conditions. We are currently developing a bioreactor instrument that maintains the environmental conditions for a vessel that grows bacteria. The system uses 16 sensors including temperature, pH, and light levels, and it uses 16 effectors to control the heat, light, and nutrients. The experiments can run 24 hours a day for 2 weeks, so humans cannot be present at all times. Our primary control programs involve maintaining and changing setpoints, recalibrating the sensors during an experiment, and diagnosing and recovering from hardware failures. We are also applying the model towards a cognitive rehabilitation tool for the assessment and treatment of frontal lobe damage. This healthcare application uses our model to simulate the generation, evaluation, and execution of daily plans for independent living.

Although the term *autonomous* is widely used, it has no formal definition and is often confused with the term automatic. It is difficult to develop and evaluate autonomous systems without a solid definition of the term "autonomous".

We have adopted a definition motivated by the clinical evaluation of brain injury patients: *Autonomy is a measure of a system's dependence on external assistance for achieving its goals.* This is the essential metric for determining whether a patient can take care of themselves and live independently after traumatic brain injury. Since no person or thing can maintain absolute independence, autonomy must be measured *relative* to a set of goals and operating conditions. As the goals and conditions become varied and complex, and involve resource constraints, then independence becomes more difficult. An autonomous system with multiple goals, resource conflicts, and potential hardware failure requires an ability to detect and correct its own errors. Autonomy is also a multidimensional property rather than a true or false proposition. When assessing the effects of traumatic brain injury, human autonomy is often described along dimensions that include the frequency that assistance is required, the conditions that cause assistance to be required, and the amount of assistance required[16].

We propose that autonomous machines be held accountable to standards in the same dimensions as humans. To be confident that our instrument can be left on its own, it should meet similar performance standards as would be applied to a human instrument operator. Thus, our model is based on the dimensions used for the clinical evaluation of human autonomy.

Neuropsychological models

A pioneer in neuropsychology, A. R. Luria, described the frontal lobes as *"a system specialized for programming, regulation and verification of activity"*[17]. Following this description, our model defines an "on-line programmer" that monitors the interactions between the external environment and the default programs. When this programmer predicts that a default program might produce an error, or if an unpredicted error actually occurs, it modifies the default program in order to handle the error. A *program* is a sequence of actions.

Based on the neuropsychological theories [20, 16, 19, 9], we have identified three types of errors that can occur with default programs. These errors are called IRRELEVANT, INEFFECTIVE, and INTERFERING programs. The neuropsychological models have also led us to identify six types of program modification that are required for human autonomy. A person must have the ability to INHIBIT, START, STOP, CONTINUE, CHOOSE, and REORDER inappropriate routines. These concepts are central elements in our model, and we will return to them later in the paper. See [14] for more detail on the connection to frontal lobe models.

In general, the frontal lobe models provide insight and data about human autonomy. They show that well learned, automatic reactions are not sufficient for autonomy (i.e., social independence). The models also contributed to our understanding that autonomous systems should be evaluated in terms of their *functional independence*. Neuropsychology also indicates that the brain uses many parallel feedback loops (sense-act circuits), instead of the segregated sensing and acting modules that are found in many AI systems. This implies that there is no centralized record of the sensory state.

The major limitation of the models is that they are informal verbal descriptions. Unfortunately, the neuropsychological models provide few details about information processing issues. In order to implement our theory and actually build autonomous instruments, we need to develop a computer model of the informal verbal descriptions.

Control system technology

Control system technology contributes well defined feedback control algorithms, such as PID and "bang-bang" methods [1], that can be used as default control programs. Adaptive and learning methods are also provided that can be used to improve the default programs with experience. Additionally, the control system methods emphasize the importance and use of mathematical models of physical processes. This perspective also suggests that encoding real-world control actions requires a more expressive action representation than is found in most AI systems. Complex real-time control software typically relies on the expressiveness of general programming constructs for hierarchical decomposition, conditional and iterative control, and variable assignment.

The major limitation of this technology is that control programs always cover a only subset of the possible operating conditions. This is due to a finite limit on the effort spent tuning, calibrating, modeling and programming the system. Due to reliance on extensive and expensive laboratory testing and tuning, most control systems are designed to operate only within a specific range of operating ranges. Assumptions about the unlikelihood of some operating conditions also place limits on the operating range. Consequently, most autonomous controllers will eventually face unexpected conditions that result in unstable behavior. When such a *program error* occurs, the control system may produce an error message, or its behavior may be undefined or dangerous. Autonomy requires a graceful degradation of behavior when faced with unprogrammed operating conditions.

A *planner* can extend the operating range of a deterministic controller by searching for and generating novel responses to novel operating conditions. This requires the integration of *feedback* control methods that take actions based on *observed* errors, with *feedforward* methods that take actions based on *predicted* errors. Planners provide search-based feedforward control by predicting program errors before they happen [4]. This type of feedforward function can been seen in Figure 1 (see next section), where the planner is labeled as the PROGRAM SIMULATOR. Adding a planning component to a control system requires extending the control system's action representation to facilitate deliberation and search. Our approach is to extend a general programming language to include nondeterministic *choice points* and the ability to predict (simulate) the effects of actions. We present this method in the next section.

AI planning methods

AI provides a variety of contributions to our theory ranging from general planning and reaction architectures to specific search algorithms. Our model is related to previous situated planning systems such as ERE[2], RoboSoar[11], XFRM[18], and PROPEL[15]. Other relevant AI methods that are outside the scope of this paper's technical focus on planning methods include fault diagnosis systems for reasoning about hardware and model failures, and learning systems for improving routine programs with experience.

Autonomy requires both *reactive* and *planning* components, and they must be well integrated. Unfortunately, it is difficult to find such integrated systems. Typically, planning systems require too many simplifying assumptions such as a static external environment, omniscient perception, unlimited planning time, and no exogenous events or variant outcomes. On the other hand, reactive models like PRS[7] are more practical because they are connected to real sensors and effectors

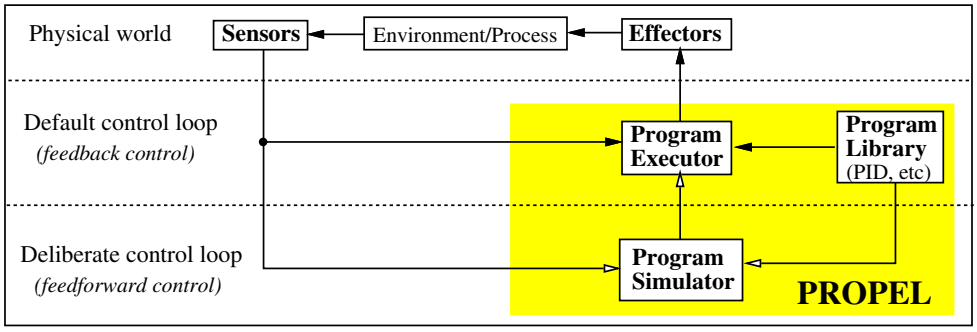

Figure 1: PROPEL overview

and can respond to external changes within guaranteed time limits. They exhibit insect level behavior using pre-programmed, circuit-like programs. However, these reactive systems cannot simulate, evaluate and modify their pre-programmed behavior. Thus they behave much like persons with frontal lobe damage.

There are many difficulties with trying to embed a planner within real-time closed-loop control software. A major source of difficulty is the fact that planning and control systems use different action representations. We call this the *language barrier*. While control systems are usually encoded in a general programming language, AI planning systems require specialized knowledge about unconventional action representations that do not use standard programming constructs for hierarchical decomposition, conditional and iterative control, and variable assignment. Another difference is that the real-time control programs deterministically select a single course of action, whereas the planning systems search through a space of many courses of actions. This language barrier makes it difficult to develop a tight integration between the planning and control modules by making it harder for the planner to understand and recover from errors that occur in the reactive control module. Another problem with using a planner in real-time control applications is that the planner's search process may not be able to produce a complete plan within a given time limit. Thus it is important for the planner to produce useful results when it is interrupted at any time[3].

Autonomy also requires both *planning* and *adaptive* control methods. Planning is required to produce an effective response on the first and perhaps only time that a given novel situation occurs. On the other hand, adaptive (learning) methods assume that the training conditions will be repeated over multiple trials. Adaptive techniques assume that, initially, some poor actions can be executed and then improved on subsequent trials, but planning is for generating a effective behavior on the first try -without any training.

This paper extends our previous work on integrated planning and control which was aimed at addressing many of the above limitations. Previously, we designed and implemented an integrated planning and reaction system called: **PROPEL - The PROgram Planning and Execution Language** [15]. This architecture, shown in Figure 1, was designed to integrate *default* and *deliberate* behavior.

PROPEL consists of a PROGRAM LIBRARY that contains nondeterministic control programs, and a PROGRAM EXECUTOR that uses heuristics to execute default programs in real-time, and a PROGRAM SIMULATOR that generates a search-space of disjunctive program instances. This provides two distinct control loops (sense-act circuits). The *default control loop* produces reactive default responses in routine situations, while the *deliberate control loop* monitors and modifies the default programs for use in novel situations. The use of two different control loops to process routine and novel responses can be found in a variety of both neuropsychological and AI models [19, 11, 6].

A primary motivation for developing PROPEL was a desire to remove the language barrier. In PROPEL, the same program definitions are used by the real-time controller (called the PROGRAM EXECUTOR), and also by planner (called the PROGRAM SIMULATOR). PROPEL's architecture is unified because the EXECUTOR and the SIMULATOR use shared data structures and algorithms for interpreting a shared action representation. This facilitates a tight integration between the PROGRAM EXECUTOR and the PROGRAM SIMULATOR that can be exploited when planning a recovery from an execution failure.

The **Program Executor** produces rapid default responses in routine situations. These default programs, called *routines*, are generated either by learning methods or by a human programmer. The routines are executed within guaranteed time limits by using heuristics to make default selections at the nondeterministic choice points that are encoded within a program.

The **Program Library** contains nondeterministic programs that map sensory conditions into effector command sequences. Each program in the library performs its own sensing in a manner similar to the brain's parallel feedback (sense-act) loops. This property is also found in some robotic systems.

The key to PROPEL's unified architecture is the *action representation* of the programs in this library. The action representation is a dialect of Lisp that uses nondeterministic *assignment statements* and *subroutine calls* to represent *choice points.* This general programming language extends the procedural expressiveness of AI action representations. The following program, called `Play-game`, illustrates PROPEL's action representation.

The tight integration between PROPEL's EXECUTOR

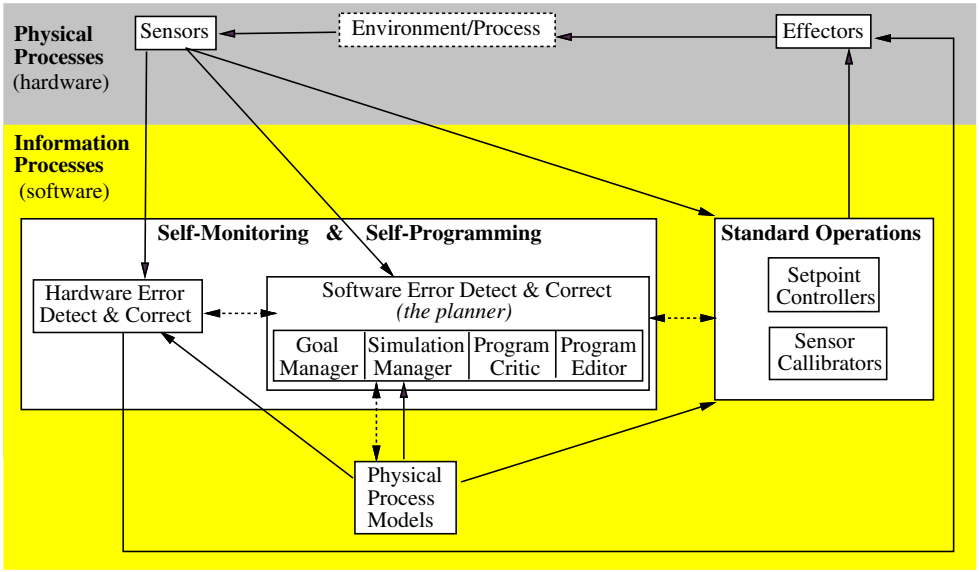

Figure 2: An autonomous control system

```
(Defprogram Play-game (board)
   :Body
   (moves ← nil)
   (Until (game-over? board)
      Do (move ← (choose-value (legal-moves board)
                               :preferences (best-moves)))
         (board ← (change-board move board))
         (push move moves))
   (print-msg "The solution is: " (reverse moves)))
```

and SIMULATOR is possible because they use identical *search engines* to interpret this dialect of nondeterministic lisp. The search tree is defined by nodes that correspond to computational processes. The root node of the tree corresponds to an initial program call, and *choice point* statements generate branches in the search tree. Each child node represents a distinct continuation of its parent, based on the selection of a different choice. The search engine functions as a scheduler that uses heuristics to bias the amount of CPU time allotted to competing continuations. PROPEL uses both local and global heuristics to bias the search. The primary difference between the two search engines is that the EXECUTER is constrained to choose only one path, but the SIMULATOR can search and backtrack through a space of many different program instances.

The **Program Simulator** generates simulations of programs in order to predict and correct upcoming errors caused by novel operating conditions. It *simulates* physical effector commands rather than actually executing them in the real world. When an error occurs in the simulation, the SIMULATOR backtracks through the search space to identify a set of non-default choices that are more effective. Simulation is only possible if planning time is available between the time the error is detected (or predicted) and the time it must be corrected. If there is no time to plan, then default behavior is the only possibility. To meet real-time deadlines, the SIMULATOR can produce usable results when it is interrupted at any time. The results are a form of Situated Control Rules[5] that advise the EXECUTOR to override default choice point selections.

The combined perspective

We now discuss a computer model, shown in Figure 2, that was designed by combining the above neuropsychological, control, and AI perspectives. Its purpose is to extend AI planning methods to perform the functions of the frontal lobe models in the context of real-time, closed-loop control applications.

We start by noting that an autonomous system is an interconnected system of physical processes *(hardware)*, with information processes *(software)*. The information processes are being implemented as PROPEL application programs. All control systems must be able to perform a set of *standard operations*, such as cooking, bathing, and shopping for humans, or maintaining a setpoint in a process control system. The PROPEL programs shown in the previous section are examples of standard operations. An autonomous system must *also* be able to detect and correct errors in its own hardware and software with minimal assistance. It is difficult to segregate the self-monitoring and self-programming functions because the hardware and software error handling modules each have their own feedback loops (sense-act cycles). These programs, and the standard operations, all respond to sensor conditions in parallel. A learning component could also provide a self-programming function, but it is not shown here because it is outside the scope of our current work.

PROPEL provides the substrate for implementing this model. This means that the software processes in Figure 2 should be viewed as PROPEL application programs. They correspond to the contents of the PROGRAM LIBRARY shown in Figure 1. Although Figure 1's EXECUTOR and SIMULATOR components are not shown in Figure 2, their presence is implicitly part of the architecture, allowing each program in Figure 2 to be either simulated or executed.

In Figure 2, the solid arrows indicate standard data flow relations between the programs. However, the dashed arrows represent "meta-links", indicating that the planner uses and modifies the *program definitions*

of the other modules as data. The figure shows that the planner must detect and correct errors not only in the standard operations programs but also in the programs for hardware fault diagnosis, and the physical process models. The figure also shows the triple role of the physical process models. These are programs that encode mathematical models of physical processes. First, these models are used to develop the standard operations programs at *design time*. Then at *run time*, the process models are used by *both* the hardware *and* the software error handling programs for simulation purposes.

Figure 2 also shows the mix of source disciplines. The setpoint control programs and the physical process models are contributions from the control system perspective. The error detection and correction modules are contributions from AI planning and diagnosis methods. The neuropsychological theories contributed in a general manner to the three component system for standard operations, self-monitoring and self-programming, and they also contributed in specific ways to the design of the planner's program CRITIC and EDITOR modules.

Our focus is on the planner, whose operations can be summarized as follows: The GOAL MANAGER maintains a dynamic list of conditions (goals), along with their associated positive and negative reward values. The SIMULATION MANAGER initiates and updates the simulation of programs in order to predict their effects. The PROGRAM CRITIC evaluates those predictions to detect potential program errors. The PROGRAM EDITOR corrects errors by modifying the program. We now describe these planning components further.

The **Goal Manager** maintains a list that associates conditions (goals), with positive or negative reward values. We call this set of weighted goals the *reward structure*. *Conditions* are also known as facts, predicates and fluents. The goal manager can ADD or REMOVE goals, or CHANGE a goal's reward value. An initial set of top level goals for safety and health are fixed at design time. New (sub)goals are then generated when the PROGRAM SIMULATOR generates hypothetical conditions that produce high rewards. Conditions with positive rewards become goals of *achievement* or *maintenance*, and those with a negative reward become goals of *prevention*.

The **Simulation Manager** initiates and updates the simulation of programs. When triggered by one of the four events listed below, it invokes PROPEL's PROGRAM SIMULATOR in order to predict the effects of a given program. The PROGRAM SIMULATOR searches through the space of disjunctive program instances, and returns a *search record* of the simulation's search space. The search record is used by the SIMULATION MANAGER to identify predicted conditions and it is also passed to the PROGRAM CRITIC.

Event	*Definition*
GOAL	Reward structure is modified
EXECUTION	Execution failure occurs
CONDITION	External conditions change
DEADLINE	Deadline approaches

A GOAL event occurs when the GOAL MANAGER modifies its reward structure. Adding new goals will trigger the initial simulation and evaluation of the default and alternative responses to that goal. Removing goals or changing their reward value will trigger dependency-directed backtracking through the planner's search space. The EXECUTION event occurs when a default program fails during execution. For example, if obstacles block the path of a mobile robot, or if it drops the coffee mug its carrying. This event also triggers dependency-directed backtracking. The CONDITION event indicates a conflict between an observed condition and a predicted condition. The predicted conditions are extracted from the search records that are produced the PROGRAM SIMULATOR. When a condition is predicted to be true, but is observed to be false, then dependency directed backtracking is again triggered. The DEADLINE event occurs when a goal's deadline moves within some threshold temporal distance. For instance, this event could trigger dependency directed backtracking with updated sensor readings 5 minutes before execution.

The **Program Critic** evaluates the SIMULATOR'S predictions to identify program errors that are caused by novel conditions. The predictions are extracted from a *search record*. Based on the frontal lobe models, we have defined the three types of program errors shown below. We can now formally define the term *novel condition* to be any condition that causes a routine to have one of the following program errors.

Error	*Definition*
IRRELEVANT	Routine preconditions do not fail, but no *active* goals are satisfied
INEFFECTIVE	Routine preconditions fail
INTERFERING	Routine causes another program to fail

The CRITIC evaluates the search record for each simulated program, and then rates the program as IRRELEVANT, INEFFECTIVE, or INTERFERING if an error is detected - or OK if there is no error. Although IRRELEVANT routines are not discussed much in AI literature, they can occur when goals are compiled out from a routine's preconditions, or if the external state is misinterpreted due to limited or imperfect perception. INEFFECTIVE routines *are* detected in AI integrated planning and reaction systems like XFRM[18], ERE[2] PROPEL[15], and Soar [11]. INTERFERING routines are interactions between conjunctive goals *in concurrent programs*. INTERFERING is a predicate that is a function of two or more programs, while INEFFECTIVE is a function of only one program.

The **Program Editor** modifies default programs to accommodate novel conditions. The table below shows some of the many ways to modify programs.

Type	*Definition*
INHIBIT	Override default *start-condition* that is TRUE
START	Override default *start-condition* that is FALSE
CONTINUE	Override default *stop-condition* that is TRUE
STOP	Override default *stop-condition* that is FALSE
CHOOSE	Override default *choice point selection*
REORDER	Override default subroutine *sequence*

The first four of these types come directly from the neuropsychological theories[16]. AI planning methods

(notably ERE and PROPEL) also motivated the START and CHOOSE types. The *start* and *stop conditions* refer to triggering preconditions and iteration termination conditions respectively.

Status and Future Work

The PROPEL substrate for our system is currently implemented as described for Figure 1. This provides our basic ability to simulate and execute PROPEL programs. The components of Figure 2, however, are still under development. Our current system supports a subset of the functionality required by the full model. Currently, the SIMULATION MANAGER responds to GOAL and EXECUTION events. The PROGRAM CRITIC can detect INEFFECTIVE routines, and the PROGRAM EDITOR can use CHOOSE and START to override default programs. Our current efforts are directed towards extending the system to support the rest of the model's full functionality. Implementing the dependency-directed backtracking used by the SIMULATION MANAGER is on the top of our agenda. We expect to build on previous dependency-directed mechanisms[22, 10] for this feature. The GOAL MANAGER also needs to be further developed.

Clearly many issues remain unresolved, and many open research questions remain unanswered, such as: Exactly how will the dependency-directed backtracking work, and will it be too slow? How is planning time and execution time allocated? How can partially ordered actions be represented and simulated? How does the model relate to natural and artificial neural nets? How are the programs learned? How can abstract programs be represented and simulated?

Conclusion

We have presented a theory of autonomous action that combines three different perspectives. The combined theory mixes the complementary contributions from each perspective, and also extends the theories from each perspective. The resulting single model has improved our understanding of the nature of autonomy. We hope this model definition provides a common language and motivation for future interdisciplinary efforts to improve upon this start.

References

[1] Bateson, R., *Introduction to Control System Technology.* Fourth Edition, MacMillan Publishing Company, New York, 1993.

[2] Bresina, J., Drummond, M. 1990. Integrating Planning and Reaction: A Preliminary Report. *Proc. of 1990 AAAI Spring Symposium (session on Planning in Uncertain, Unpredictable, or Changing Environments)*, Stanford, CA.

[3] Dean, T., and Boddy, M. An Analysis of Time-Dependent Planning. *In Proc. of AAAI-88*, pp. 49–54, St. Paul, MN, 1988.

[4] Dean, T. and Wellman, M. *Planning and Control.* Morgan Kaufman Publishers, San Mateo, CA. 1991.

[5] Drummond, M. Situated Control Rules. *Proc. of the First International Conference on Principles of Knowledge Representation and Reasoning*, (KR-89) Toronto, Canada, 1989.

[6] Drummond, M., R., Bresina, J., Swanson, K. and Levinson, R.
Reaction-First Search: Incremental Planning with Guaranteed Performance Improvement. *The proceedings of IJCAI-93.* Chambery, France. 1993.

[7] Georgeff, M. P. and Lansky, A.L., Reactive Reasoning and Planning, in: *Proceedings of AAAI-87*, Seattle, WA (1987) 677-682.

[8] Grafman J, and Hendler, J. Planning and the brain. *Behav. Brain Science.* 1991;14:563-564.

[9] Grafman, J., Sirigu, A., Spector, L, Hendler, J., Damage to the prefrontal cortex leads to decomposition of structured event complexes. in *Journal of Head Trauma Rehabilitation*, Vol. 8, Number 1, March 1993.

[10] Kambhampati, S. Mapping and Retrieval during Plan Reuse: A Validation Structure Based Approach. *Proceedings from AAAI '90*, Boston, MA. 1990

[11] Laird, J. and Rosenbloom, P. Integrating Execution, Planning and Learning in Soar for external environments, *Proceedings of AAAI-90*, Boston, MA. 1990.

[12] Levinson, R., Robinson, P., and Thompson, D. Integrated Perception, Planning and Control for Autonomous Soil Analysis. *Proceedings of the 9th IEEE Conference on AI for Applications*, Orlando, FL. 1993.

[13] Levinson, R., Human Frontal Lobes and AI Planning Systems, in: *Proceedings of the Second International Conference on AI Planning Systems (AIPS-94)* Chicago, IL. 1994.

[14] Levinson, R. A Computer Model of Human Frontal Lobe Function (A Preliminary Report). NASA Ames Research Center, AI Research Branch Technical Report FIA-94-15. September 1994.

[15] Levinson, R., A General Programming Language for Unified Planning and Control. *Artificial Intelligence* special issue on Planning and Scheduling. To appear 1995.

[16] Lezak, M. *Neuropsychological Assessment.* Oxford Univ. Press. New York, 1983.

[17] Luria, A. R., *The Working Brain.* Basic Books Inc. New York. 1973.

[18] McDermott, D., Transformational Planning of Reactive Behavior. Yale University Department of Computer Science, Technical Report YALEU/CSD/RR#941, December, 1992

[19] Shallice, T., Burgess, P., Higher-Order Cognitive Impairments and Frontal Lobe Lesions in Man. In *Frontal Lobe Function and Dysfunction*, edited by H. Levin, H. Eisenberg, and A. Benton. Oxford University Press, 1991.

[20] Sohlberg, M., Mateer, C., *Introduction to Cognitive Rehabilitation.* Guilford Press, New York, 1989.

[21] Spector, L., Grafman, J. Planning, Neuropsychology, and Artificial Intelligence: Cross-fertilization. in *Handbook of Neuropsychology*, edited by Grafman. 1994.

[22] R. Zabih, D. McAllester and D. Chapman, Nondeterministic Lisp with Dependency-Directed Backtracking, in: *Proc of AAAI '87*, Seattle, WA. 1987