Monk TH (1991) *Sleep, Sleepiness and Performance*. New York, NY: John Wiley.
Pilcher JJ and Huffcut AJ (1996) Effects of sleep deprivation on performance: a meta-analysis. *Sleep* **19**: 318–326.
Pinel JPJ (2003) *Biopsychology*, 5th edn. Boston, MA: Allyn & Bacon.
Stampi C (1992) *Why we Nap: Evolution, Chronobiology and Functions of Polyphasic and Ultrashort Sleep*. Boston, MA: Birkhauser.
Van Cauter E and Copinschi G (2000) Interrelationships between growth hormone and sleep. *Growth Hormone IGF Research* **10**(Supplement B): S57–62.
VanHelder T and Radomski MW (1989) Sleep deprivation and the effect on exercise performance. *Sports Medicine* **7**: 235–247.
Wesensten NJ, Balkin TJ and Belenky G (2000) Does sleep fragmentation impact on recuperation? A review and re-analysis. *Journal of Sleep Research* **8**: 237–245.

Ritter, F. E. (2003). Soar. In L. Nadel (Ed.), Encyclopedia of cognitive science. vol. 4, 60-65. London: Nature Publishing Group. [A006.pdf]

Soar

Intermediate article

Frank E Ritter, Pennsylvania State University, Pennsylvania, USA

CONTENTS

Soar is a unified theory of cognition, and a cognitive architecture, realized as a production system, a type of expert system. It is designed to model human behaviour on multiple levels.

UNIFYING COMPUTATIONAL MECHANISMS TO FORM A THEORY OF COGNITION

Soar is a unified theory of cognition (UTC) realized as a computer program. It can be considered in three mutually complementary ways. First, it can be seen as a theory of cognition realized as a set of principles and constraints on cognitive processing: a cognitive architecture (Newell, 1990). In this respect, Soar provides a conceptual framework for creating models of how people perform tasks, typically assisted by the corresponding computer program. In this view Soar can be considered as an integrated architecture for knowledge-based problem solving, learning, and interacting with external environments. It is thus similar to other unified theories in psychology, such as ACT-R, EPIC, PSI, and CAPS. (*See* **Skill Acquisition: Models**; **Learning and Memory, Models of**)

Second, Soar can be seen as the computer program that realizes a particular theory of cognition. There are debates as to whether and how the theory is different from the computer program, but it is fair to say that they are at least highly related. It is generally acknowledged that the program implements the theory, and also that there are commitments that are not in the theory but that must be made in the program to create a running system. In this way it is similar to other cognitive theories realized as computer programs, such as ACT-R, and connectionist models of specific tasks realized as programs.

Third, Soar can be seen simply as a specialized AI programming language. In this view, what matters is only that it performs the task in an intelligent way. In this respect it is similar to expert system tools such as OPS5 and CLIPS. (*See* **Expert Systems**)

The deliberate combination of these approaches to understand intelligence has been fruitful. Researchers interested in creating cognitive models have used Soar primarily to model human behaviour in detail, to suggest new uses of existing mechanisms to create behaviour, and to propose

improvements to the Soar programming interface. Researchers interested in creating AI programs have contributed to the efficiency, functionality, and generality of Soar as a programming language and provided information on the functional requirements of working systems.

SOAR AS A UNIFIED THEORY OF COGNITION

Soar was proposed by Newell (1990) as a candidate UTC. Newell presents a full description of the virtues of unification. Three of the most important are: coherence in theorizing ('it is one mind that minds it all'); bringing to bear multiple constraints from empirical data; and reducing the number of theoretical degrees of freedom. (*See* **Unified Theories of Cognition**)

Being a UTC does not mean that there is only a single mechanism for each task or behaviour, although in most places in Soar there is only one. It does mean that the set of unifying principles and mechanisms must work together to support all of cognition: there is not a big switch or a set of disjoint modules (Newell, 1992). ACT-R is another unified theory, although the set of mechanisms it proposes to account for all of human behaviour is longer. (*See* **ACT**)

Unified theories represent a grand vision. None of them can yet provide even a verbal explanation for all of human behavior in terms of architectural mechanisms, let alone provide implemented models, and few have yet covered more than a small set of regularities. This name may even be a misnomer, for they are attempting to unify all of behavior, not just cognition. Their intention is to cover larger amounts of data than have been covered before, and to bind the different areas of cognition together through a common set of mechanisms. A common and unproductive criticism is that an architecture is wrong because all areas are not yet covered. All theories suffer from this limitation. A much more valid and valuable criticism would be that an important aspect of a given area cannot be accounted for by the current architecture.

GOAL-DIRECTED SEARCH IN HIERARCHICAL PROBLEM SPACES BASED ON PRODUCTION RULES

Soar – as a theory, as a cognitive modeling language, and as an AI programming language – incorporates problem spaces as a single framework for all tasks and subtasks to be solved, production rules as the single representation of permanent knowledge, objects with attributes and values as the single representation of temporary knowledge, automatic subgoaling as the single mechanism for generating goals, and chunking as the single learning mechanism. Specifically, Soar provides a general scheme for control – deciding what to do next – that is hypothesized to apply to all cognition. These mechanisms can be used in different ways, however. For example, chunking can be used to learn both declarative and procedural knowledge.

Soar can be viewed at three levels. At the highest level, it approximates a knowledge-level system (Newell, 1982). This is an abstract level where a system is described in terms of its knowledge, and which is only approximated by any realized system, including Soar. The two lower levels are the problem space level and the symbol level. These work together to support learning.

The Problem Space Level

Figure 1 illustrates the two lower levels. These are similar to Marr's lower two level of analysis. The higher of these levels is the problem space level, where behaviour is seen as occurring in a problem space made up of goals, problem spaces, states, and operators. Note that these terms refer to specialized constructs in Soar, which are related to, but not strictly equivalent to, their usual meanings in

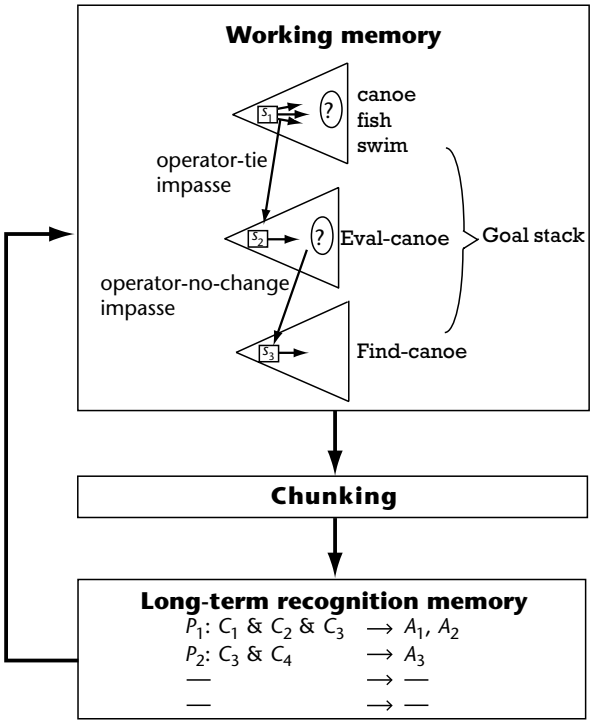

Figure 1. Structures in Soar.

cognitive science. (*See* **Computer Modeling of Cognition: Levels of Analysis**)

A problem space is a set of representations for a problem, the structures for states, and all the operators relevant to those representations. The operators may be implicit and shared with other spaces. There can be several problem spaces active at any one time. A state may lack some required knowledge and have a state created to help it find the knowledge it needs, and similarly be providing knowledge itself. In figure 1 this state relationship is shown in the states S_1, S_2, and S_3. The main reason for organizing knowledge into problem spaces is that it reduces the search for information. This approach has also been used successfully as a software design technique.

While solving a problem there is a current state structure that specifies the situation of the problem solver in each problem space. For example, in a blocks world, the state might be 'block A is on top of block B, and block B is on the table'.

Fluent, expert behavior consists of a repeated cycle on the problem space level in which an operator is selected and is applied to the current state to produce a new (modified) current state. The process of choosing and applying a new operator (or creating a new state) is called a decision cycle. So in the example above, we could have applied an operator to move block A onto the table, after which the current state would include the fact that block A and block B are both on the table.

The Symbol Level

The problem space level is realized by a lower level, a symbol level. At this level of analysis long-term recognition memory, realized as production rules, is compared to the current set of contexts. Rules (shown as P_1 and P_2 in figure 1) will have their conditions (C_1, C_2, etc.) matched to the current context. Their actions (A_1, A_2, etc.) will act on the problem space level to generate operators, propose how to choose between operators, implement operators, or augment the state with known inferences. Each cycle of rule application is called an elaboration cycle. There may be several elaboration cycles in each decision cycle. All rules whose conditions are satisfied are allowed to apply. If they make conflicting suggestions, the architecture sorts them out using an impasse (see below).

The rules are structured to match objects in the architecture. The rules can test the contents of states, and test for operators by name and by their contents. The rules' outputs are constrained to be in terms of the problem space structures. These constraints on the representation of the rules are part of what makes the system a cognitive architecture and not simply a free-form programming language.

Soar uses a modified RETE algorithm to apply the production rules. The time this algorithm takes to match a rule set is proportional to the number of memory elements that change, not the number of rules. This leads to very little slowdown as larger rule sets are used (but requires more computer memory). The largest systems created have had over a million rules with little or no slow down with additional rules (Doorenbos, 1995; Doorenbos *et al*., 1992).

Learning and Chunking

But what happens if something prevents the process of operator application from continuing smoothly? For example, perhaps the current knowledge in the Soar model cannot propose any operators to apply to the current state. Or the model may know of several applicable operators, but has no knowledge of how to choose between them. In such cases, the Soar model encounters an *impasse*. There is a limited number of types of impasse defined by the architecture, which primarily arise through a lack of knowledge (inability to apply or select an operator) or through inconsistent knowledge (conflict among operator proposals).

When Soar encounters an impasse in context level 1, it sets up a subcontext, a subgoal, at level 2, which has associated with it a new state, which may end up with its own problem space and operators. Note that the operators at level 2 could well depend upon the context at level 1. The goal of the level 2 context is to find knowledge sufficient to resolve the higher impasse, allowing processing to resume there. For example, we may not have been able to choose between two operators, so the level 2 subgoal may simply try one operator to see if it solves the problem, and if not, try the other operator.

The processing at level 2 might itself encounter an impasse, set up a subgoal at level 3, and so on. The problem solver usually has a stack of such levels, each generated by an impasse in the level above. Each level can have its own state, problem space, and operators.

In Figure 1, there were several operators proposed for the pond, including canoeing and fishing, and no knowledge was available to choose between them, so a new context was created to allow the architecture to consider this problem explicitly in a selection problem space, through what is called an 'operator-tie impasse'. Knowledge was

available in that space, which suggested testing the canoeing operator and seeing how it would play out. The operator `Eval-canoe` was attempted, but nothing happened, so another impasse (an 'operator-no-change' impasse) was declared and an operator could be proposed in an evaluation problem space.

Whenever processing in the subgoal generates results that allow a higher level to continue, for example, if the operator `Find-canoe` allows `Eval-canoe` to continue, the architecture notices this, and automatically generates a new rule (also called a chunk) to summarize this problem solving. This rule's conditions are based on backtracking through the problem solving to find out what aspects of the initial situation were used, and the rule's actions are the output of `Find-canoe` that removed the higher-level impasse. In this case it would probably be a change to the `Eval-canoe` operator or to its state.

The next time such a condition occurs, the rule will match and update the operator or state, and the impasse will be avoided. This is the basic learning mechanism in Soar. This approach provides a strong theory of when and how learning and transfer will occur.

Chunking has been used to create a wide range of higher-level learning – including explanation-based learning, declarative learning, instruction taking, and proceduralization – by varying the type of impasse and the knowledge used to resolve it.

THE HISTORY OF SOAR

The intellectual origins of Soar can be found in the seminal work of Newell and Simon on human problem solving. This builds upon work on production system architectures in the 1970s onwards, particularly Newell's work on the problem space as a fundamental category of cognition (Newell, 1980). Soar as a unified theory of cognition has some of its theoretical roots in the Model Human Processor (Card *et al*., 1983).

The first implementation of Soar was built by Laird, modifying Rosenbloom's XAPS architecture. Impasses were introduced in Soar 2, a reimplementation of Soar in OPS5, which allowed rules to fire in parallel and included the problem space decision mechanism. The original motivation was both functional (to create an architecture that could support problem solving using many different weak methods arising from the knowledge that was available) and structural (to create an architecture that integrated problem spaces and production systems). 'SOAR' was originally an acronym for 'state operator and result', but it is no longer recognised as being an acronym because the theory is more complex.

A major watershed in the development of Soar was Newell's William James lectures at Harvard (Newell, 1990). These lectures defined what a unified theory in psychology should include, proposed Soar as a candidate unified theory, and extended the Soar theory, providing some detailed examples. (*See* **Newell, Allen**)

The recent development of Soar has been driven by applications. Soar models have been applied to real-time domains such as flying simulated aircraft (Jones *et al*., 1999). Analyses of running models showed that the state and problem space in the original Soar theory were not being used as had been initially imagined: in most cases they did not vary and were simply reiterations of the goal. Later versions of Soar have dropped problem spaces and states as explicit reserved structures but allowed the modeller to represent them in the goal. This has led to faster systems that allow several models on a single computer to interact in real time, performing complex tasks. Because these context slots were not being used by models, their removal did not lead to changes in behavior.

Architectural work on Soar is currently focused on improving its interface, introducing new learning algorithms built upon the chunking mechanisms, tying Soar to external worlds (including behaviour moderators like stress), and the implications of interaction for problem solving and learning. Future work could include reviving the Neuro-Soar project (Cho *et al*., 1991). This project showed that it was possible to realize the symbol level of Soar with a connectionist network, although modelling so many theoretical levels made it slow.

MATCHING HUMAN PERFORMANCE IN DIVERSE DOMAINS

One of the strengths of Soar is that it predicts the action sequences and times to perform tasks (Newell, 1990). The parameters chosen by Newell have been gradually refined. The Soar philosophy has been to retain the same constraints from problem to problem, rather than having numerous parameters that can be adjusted for a specific task or data set.

For cognitive modelling, Soar is most effective at modelling deliberate cognitive human behavior at timescales greater than 50 ms. Published models include human–computer interaction tasks, typing, arithmetic, categorization, video game playing (i.e., rapid interaction), natural language understanding,

concept acquisition, learning by instruction, verbal reasoning, driving, job shop scheduling, and teamwork.

Soar has also been used for modelling learning in many of these tasks, many of which involve interaction with external environments. Soar does not yet have a standard model for low-level perception or motor control, but two systems that could be integrated, EPIC-Soar (Chong and Laird, 1997) and Sim-eyes and Sim-hands (Ritter *et al*., 2000), have been created. Learning adds significant complexity to the structure of the model, however. (*See* **Learning Rules and Productions**)

One of the signature data regularities modeled in Soar is the learning curve. The learning curve predicts that the time to do a task decreases according to a power law (or perhaps an exponential decay). Soar's prediction of the power law of practice for a task arises from how models in Soar do the task and what they learn.

The first, and probably the simplest, way in which the power law of learning has been modelled in Soar is for the Seibel task. This simple task is to push the buttons on a panel corresponding to lights that are on. There are ten lights, therefore 1023 possible patterns of lights where at least one light is on. The model proposes two operators to do a left and a right subregion. If these are not individual lights, then an impasse occurs, and each subregion gets two operators. This continues until a single light is a subregion. The model can then return a chunk that does both subregions, initially, two lights. Early trials generate two-light patterns that occur often and are useful. Later trials can build larger patterns, with more lights, that occur less often but save more time.

The Seibel model was one of the first learning models in Soar, and represents probably the simplest approach to learning in Soar. It does not represent more complex and accurate learning methods. Current models include learning by instruction, learning by following others, modeling transfer between tasks, and learning category knowledge. We are now at the point where, if we can model performance on a task in Soar, we expect to be able to model learning. Nearly all of the cognitive models in Soar are models that learn, and a majority of these have been compared with data.

SOAR AS AN EXPERT SYSTEM DEVELOPMENT ENVIRONMENT

Soar has also been used to create a variety of classification expert systems, that is, systems that classify situations. These including lift planning, production scheduling, diagnosis, robotic control, and computer configuration. It has been used in the Sisyphus knowledge elicitation comparisons.

Perhaps the greatest success for Soar expert systems has been in a procedural domain, flying simulated aircraft in a simulated hostile military environment. In one experiment Soar flew all of the US aircraft in an international 48-hour simulation exercise (Jones *et al*., 1999). The simulated pilots talked with each other and with ground control, and carried out over 700 sorties with up to 100 planes in the air at once.

For building artificial intelligence (AI) and expert systems Soar's strengths are in: integrating knowledge; planning; the ability to react quickly by modifying its internal state or changing its goal stack; search; and learning within a very efficient architecture. It also has the ability, used in a model that plays Quake®, to create a state mirroring its opponent's state, and consider what the opponent will do by considering what it would do itself in the same situation.

CHALLENGES FOR SOAR AND OTHER UTCs

Like any unified theory of cognition realized as a program, Soar faces major challenges. Work continues on applying Soar to a wider range of tasks and including learning and interaction in these models. Meanwhile, usability is becoming increasingly important as Soar moves out of the academic world into the world at large.

Soar has been developed and used by a community of researchers. Keeping a group of up to 100 researchers together intellectually has been difficult. Explicit mechanisms are necessary, such as repositories of papers and programs, regular meetings, mailing lists, frequently asked question lists (FAQs), and websites.

SUMMARY

There are a number of relatively unique capabilities that arise out of the combination of the structures and mechanisms in Soar. First, problem solving and learning are tightly intertwined: chunking depends on the problem solving, and most problem solving would not work without chunking. Secondly, interruptibility is available as a core aspect of behaviour. Rules are matched against the whole context stack. Processing can thus proceed in parallel on several levels. If the situation changes, rules can fire quickly, suggesting new operators at the level most appropriate for dealing

with the change. Thirdly, it is possible to create large rule systems because they can be organized in problem spaces; and the architecture makes them fast to build and to run. Fourthly, planning can be integrated with reacting as well as with dynamic decomposition of tasks.

It takes effort to learn Soar. More practice is needed than for other, simpler, systems. Those projects that have used Soar successfully have often been able to solve problems that were previously unsolvable or unmodellable, but not without hard work on the part of the modellers.

Soar is, perhaps uniquely, appropriate for creating large cognitive models or expert systems, or for projects where learning or interaction (or both) are important.

References

Card S, Moran T and Newell A (1983) *The Psychology of Human-Computer Interaction*. Hillsdale, NJ: Erlbaum.

Cho B, Rosenbloom PS and Dolan CP (1991) Neuro-Soar: a neural-network architecture for goal-oriented behavior. In: *Proceedings of the 13th Annual Conference of the Cognitive Science Society*, pp. 673–677. Hillsdale, NJ: Erlbaum.

Chong RS and Laird JE (1997) Identifying dual-task executive process knowledge using EPIC-Soar. In: *Proceedings of the 19th Annual Conference of the Cognitive Science Society*, pp. 107–112. Mahwah, NJ: Erlbaum.

Doorenbos R, Tambe M and Newell A (1992) Learning 10,000 chunks: what's it like out there? In: *Tenth National Conference on Artificial Intelligence (AAAI'92)*, pp. 830–836. Menlo Park, CA: AAAI.

Doorenbos RB (1995) *Production Matching for Large Learning Systems*. PhD thesis, Carnegie-Mellon University. Tech. Report CMU-C5-95-113.

Jones RM, Laird JE, Nielsen PE *et al*. (1999) Automated intelligent pilots for combat flight simulation. *AI Magazine* **20**(1): 27–41.

Newell A (1980) Reasoning, problem solving and decision processes: The problem space as a fundamental category. In: Nickerson RS (ed.) *Attention and Performance VIII*. Hillsdale, NJ: Erlbaum.

Newell A (1982) The knowledge level. *Artificial Intelligence* **18**: 87–127.

Newell A (1990) *Unified Theories of Cognition*. Cambridge, MA: Harvard University Press. [Précis: Unified theories of cognition. *Behavioral and Brain Sciences* **15**: 425–492.]

Ritter FE, Baxter GD, Jones G and Young RM (2000) Supporting cognitive models as users. *ACM Transactions on Computer-Human Interaction* **7**(2): 141–173.

Further Reading

Laird JE and Rosenbloom PS (1992) In pursuit of mind: the research of Allen Newell. *AI Magazine* **13**(4): 17–45.

Laird JE and Rosenbloom PS (1995) The evolution of the Soar cognitive architecture. In: Steier DM and Mitchell TM (eds) *Mind Matters*, pp 1–50. Hillsdale, NJ: Erlbaum.

Rosenbloom PS, Laird JE and Newell A (1992) *The Soar Papers: Research on Integrated Intelligence*, 2 vols. Cambridge, MA: MIT Press.

Ritter FE, Baxter GD, Avraamides M and Wood AB (2001) *Soar FAQ*. [http://ritter.ist.psu.edu/soar-faq.html.]

The Soar Group: http://ai.eecs.umich.edu/soar/soar-group.html.