**Soar**

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

[really 2003, not 2002]

ENCYCLOPAEDIA OF COGNITIVE SCIENCE

September 2001, published in 2002

'Macmillan Reference Ltd

Article Reference Code: 6

Article Title: Soar

Level of article (see below): Level 2

**Keywords** unified theory of cognition#power law of learning#cognitive models and modelling#learning#expert system#

Ritter, Frank E

Frank E Ritter

The Pennsylvania State University, University Park, Pennsylvania USA

Approximately 3750 words + 1 figure + 11 references

Full contact details:

Frank E. Ritter
School of Information Sciences and Technology
Thomas Building Basement
Penn State
University Park, PA 16802

## Contents list

1. Unifying computational mechanisms to form a theory of cognition

2. Soar as a Unified Theory of Cognition

3. The details: Goal-directed search in hierarchical problem spaces based on production rules

    3.1 The problem space level

    3.2 The symbol level

    3.3 Learning and chunking -- What makes Soar special

4. The history of Soar

5. Matching human performance in diverse domains

6. Soar as an expert system development environment

7. Challenges for Soar and other UTCs

8. Summary

## Very short summary

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

## 1. Unifying computational mechanisms to form a theory of cognition

Soar is a unified theory of cognition (UTC) realized as a computer program. Soar can be considered in three complementary ways. First, Soar 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. [also cite other relevant sections in encyclopedia].

Second, Soar can be seen as the computer program that realizes that 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 that there are commitments in the program that must be made to create a running system that are not in the theory -- places where the current theory does not say one thing or another. In this way it is similar to other psychology theories realized as computer programs, such as ACT-R and individual connectionist models of specific tasks realized as programs. [also cite relevant sections in encyclopedia].

Third, Soar can also be seen as simply as a specialized AI programming language. In this view only performing the task in an intelligent way is important. In this way, it

is similar to expert system tools such as OPS5 and CLIPS. [also cite relevant sections in encyclopedia, e.g., expert systems and examplars like Mycin].

The deliberate combination of these approaches to understand intelligence by confounding research on AI and psychology has been fruitful. Researchers interested in creating cognitive models have used Soar primarily to model human behavior in detail, to suggest new uses of existing mechanisms to create behavior, 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 building working systems.

## 2. Soar as a Unified Theory of Cognition

Soar was put forward by Newell (1990) as a candidate UTC. Newell's book presents the full description of the virtues of unification. Three of the most important include (a) coherence in theorizing: 'It is one mind that minds it all'. (b) Bringing to bear multiple constraints from empirical data. (c) Reducing the theoretical degrees of freedom.

Being a unified theory of cognition does not mean that there is only a single mechanism for each task or behavior, although in most places in Soar there is only one. It does mean that the set of unifying principles and mechanisms are required to work together as a single set to support all of cognition -- there is not a big switch or a set of disjoint modules. This point has often been misunderstood, but has received some support and discussion, for example, in a target article in *Brain and Behavioral Sciences* (Newell, 1992). ACT-R is another unified theory in that it proposes a larger set of mechanisms but is still a fixed set designed to account for all of human behavior.

Unified theories represent a grand vision. None of them can yet provide even a verbal explanation referencing architectural mechanisms to explain all of human behavior let alone provide implemented models, and few have yet covered more than a small set of regularities. (The name may even be a misnomer, for they are attempting to be unified theories of all human behavior, not just cognition.) The intention of these endeavors is to cover larger amounts of data than have been attempted before, and to keep the subareas tied together through a common set of mechanisms, which they do. A common and unproductive criticism is that an architecture is wrong because all areas are not yet covered. All theories suffer from this limitation. Newell (1990, p. 38, citing McCulloch) noted 'Don't bite my finger, look where I'm pointing'. While all areas are not yet covered (but should be), a much more valid and valuable criticism would be that an important subarea cannot be accounted for by the current architecture.

## 3. The details: Goal-directed search in hierarchical problem spaces based on production rules

Soar, as a theory, as a cognitive modelling 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 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. It has two lower levels, the problem space and symbol level. They work together to support learning.

### 3.1 The problem space level

Figure 1 shows a graphic depiction of the two lower levels. These two are similar to Marr's lower two level of analysis **[see entry this encyclopedia]**. The higher of these levels is the problem space level where behavior is seen as occurring in a problem space, made up of Goals, Problem Spaces (PS), States (S) and Operators (O or Op). Note that these terms here name specialized constructs in Soar, and are related to, but not strictly equivalent to, their typical meanings in cognitive science.

A problem space is a set of representations for a problem, the structures for states, and all the operators relevant to that representation. The operators may be implicit and shared with other spaces as well. There can be several problem spaces active at any one time. Each 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 depicted through the states S1, S2, and S3. The main idea behind organizing knowledge into problem spaces is that it reduces the search for information. This approach has also been used as a successful software design technique.

While problem solving 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 consist of '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 (i.e., modified) current state. The process of choosing and applying a new operator

(or creating a new state) is called a decision cycle. So in our previous example, we could have applied an operator to move block A to the table, in which case the current state would include that block A and block B are both on the table.

### 3.2 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 P1 and P2 in Figure 1) will have their conditions (e.g., C1, C2) matched against the current context. Their actions (e.g., A1, A2) 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, and there may be multiple elaboration cycles per decision cycle. All rules that match are allowed to apply. If they make conflicting suggestions, the architecture sorts them out using an impasse.

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

Soar uses a modified RETE algorithm to apply the production rules. This algorithm takes time to match a rule set proportional to the number of memory elements that change, not the number of rules. This leads to very little slow down as larger rule sets are used (but requires larger computer memories). The largest systems created

have had over a million rules with little or no slowdowns with additional rules (Doorenbos, 1995; Doorenbos, Tambe, and Newell, 1992).

### 3.3 Learning and chunking -- What makes Soar special

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 that state. Or the model knows of several 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 impasses defined by the architecture, which primarily arise through a lack of knowledge (inability to apply or select an operator) or through inconsistent knowledge (conflict in the operator choice).

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, tries 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 typically 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-canoeing was attempted, but nothing happened (an operator no-change), so another impasse was declared and an operator could be proposed in an evaluation problem space.

[Insert figure 1 about here]

Whenever processing in the subgoal generates results that allow a higher level to continue, such as 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 action(s) are the output of Find-canoe that removed the higher level impasse. In this case it would likely be a change to the Eval-canoe operator or to its state.

The next time that such a condition occurs, the rule will match and update the operator or state, and the impasse would 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 by varying the type of impasse and the knowledge used to resolve it, including explanation-based learning, declarative learning, instruction taking, and proceduralization.

## 4. The history of Soar

The intellectual origins of Soar can be found in the seminal work of Newell and Simon on human problem solving. It builds upon the work on production system architectures from the 1970s onwards -- Newell's work on the problem space as a fundamental category of cognition (Newell, 1980b). Soar as a unified theory of cognition also draws some of its theoretical roots from the Model Human Processor envisioned by Card, Moran, and Newell (1983).

The first implementation of Soar was done 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 two-fold: (a) Functionality: Create an architecture that could support problem solving using many different weak methods where the method arose based on the knowledge that was available. (b) Structurally: Create an architecture that integrated problem spaces and production systems. Soar initially was an acronym, State Operator And Result, but it is no longer recognized as being an acronym because the theory is more complex.

A major watershed in the development of Soar was when Newell gave the William James lectures at Harvard. These lectures provide a platform to summarize major work in psychology. Newell used them to define what a unified theory in psychology should include, provided Soar as a candidate unified theory, and extended the Soar theory providing some detailed examples. These lectures were later turned into the 'UTC' book (Newell, 1990).

In the last decade Soar's development has been driven by applications. Soar models have been applied to real-time domains such as flying simulated aircraft (Jones, Laird, Nielsen, Coulter, Kenny, and Koss, 1999). Analyses of running models showed that the state and problem space in the original Soar theory were not being

used as 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/programmer to represent them on the goal. These steps have led to faster systems that allow multiple 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, new learning algorithms built upon the chunking mechanisms, tying Soar to external worlds, including behavior moderators like stress, and the implications of interaction for problem solving and learning. Future work could include reviving the Neuro-Soar project (Cho, Rosenbloom, and Dolan, 1991). This project showed it was possible to realize the symbol level of Soar with a connectionist network, although modeling so many theoretical levels made it quite slow. Further information on the history of Soar is available in Laird and Rosenbloom (1992; 1995), and implicitly in Rosenbloom, Laird, and Newell (1992).

## 5. 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) explains this in detail. Newell's numbers have been refined over the last ten years. The Soar philosophy has been to stick with constraints from problem to problem and not to have numerous parameters that can be adjusted for a specific task or data set.

For cognitive modeling, Soar's strengths are in modeling deliberate cognitive human behavior at time scales 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 (the Soar FAQ provides citations to these and more).

Soar has also been used for modeling learning in many of these tasks. Learning adds significant complexity to the structuring of the model, however, and is not for the casual user. Many of these tasks 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.

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 arises out of how models in Soar do the task and what they learn.

The first way, and probably the simplest way the power law of learning has been modeled 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, leading to 1023 different patterns of lights where at least one light is on. The model proposes two operators to do a left and 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 quite often and are very 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 the

current, more complex and accurate learning methods. Current models include learning by instruction, learning by following others, modeling transfer between tasks, and learning category knowledge. Now we are at the point where, if we can model performance on a task in Soar, we expect to be able to model learning (cf. position in cognitive science until just recently). Nearly all of the cognitive models in Soar are models that learn, and a majority of these have been compared with data.

## 6. Soar as an expert system development environment

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

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

For building 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.

## 7. Challenges for Soar and other UTCs

As one of the first unified theories of cognition realized as a program, Soar has faced and still faces several challenges. These challenges will also apply to other unified

theories of cognition. Work continues on applying Soar to a wider range of tasks and including learning and interaction in these models.

Newell (1990, p. 508) wrote that science, more than politics, is the art of the possible. Usability has become increasingly recognized as important. As Soar moves out of the universities that created it, it must be usable by more casual users with less formal training.

The process of developing Soar has required the use of a community of researchers. Keeping a group of up to 100 researchers together intellectually has been difficult. Explicit mechanisms are necessary, such as papers and programs repositories, yearly meetings, mailing lists, Frequently Asked Questions (FAQs) lists, and web sites.

## 8. Summary

There are a number of relatively unique capabilities that arise out of the combination of the structures and mechanisms in Soar. (a) Problem solving and learning are tightly intertwined: chunking depends on the problem solving, and most problem solving would not work without chunking. (b) Interruptibility is available as a core aspect of behavior. Rules match 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; (c) 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. (d) Planning can be integrated with reacting as well with dynamic decomposition of tasks.

It takes effort to learn Soar. More practice is needed than for other, simpler, systems. The projects that have used Soar successfully have often been able to

solve problems that were previously unsolvable or unmodellable, but Soar has not presented a magic bullet.

If you want to create a large cognitive model or an expert system, then Soar is quite appropriate. Also, Soar is appropriate for projects where learning, interaction, or both are important. If these capabilities are needed to model human data or to perform a task as an expert system, then Soar may not only be just what is needed, but may also be the only system available.

See also: Unified theories of cognition, ACT-R, power law of learning, cognitive modeling; others based on sections in the encyclopedia. computational learning theory#computational models: why build them?#computer modeling of cognition: levels of analysis#history of cognitive science and computational modelling#learning rules and productions#production systems and rule-based inference#skill acquisition: models#Unified theories of cognition

## References

1. Card, S., Moran, T., and Newell, A. (1983). *The psychology of human-computer interaction*. Hillsdale, NJ: Lawrence Erlbaum.

2. Cho, B., Rosenbloom, P. S., and Dolan, C. P. (1991). Neuro-Soar: A neural-network architecture for goal-oriented behavior. In *Proceedings of the 13th Annual Conference of the Cognitive Science Society*. 673-677. Hillsdale, NJ: Lawrence Erlbaum.

3. Chong, R. S., and Laird, J. E. (1997). Identifying dual-task executive process knowledge using EPIC-Soar. In *Proceedings of the 19th Annual Conference of the Cognitive Science Society*. 107-112. Mahwah, NJ: Lawrence Erlbaum.

4. 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)*. 830-836.

5. Doorenbos, R. B. (1995). *Production matching for large learning systems*. PhD, Computer Science (Tech. Report CMU-CS-96-167), Carnegie-Mellon University.

6. Jones, R. M., Laird, J. E., Nielsen, P. E., Coulter, K. J., Kenny, P., and Koss, F. V. (1999). Automated intelligent pilots for combat flight simulation. *AI Magazine, 20*(1), 27-41.

7. Newell, A. (1980b). Reasoning, problem solving and decision processes: The problem space as a fundamental category. In R. Nickerson (Ed.), *Attention and performance VIII*. Hillsdale, NJ: Lawrence Erlbaum.

8. Newell, A. (1982). The knowledge level. *Artificial Intelligence, 18*, 87-127.

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

10. Newell, A. (1992). Précis of *Unified theories of cognition*. *Behavioral and Brain Sciences, 15*, 425-492.

11. Ritter, F. E., Baxter, G. D., Jones, G., and Young, R. M. (2000). Supporting cognitive models as users. *ACM Transactions on Computer-Human Interaction, 7*(2), 141-173.

## Further reading

1. Laird, J. E., and Rosenbloom, P. S. (1992). In pursuit of mind: The research of Allen Newell. *AI Magazine, 13*(4), 17-45.

2. Laird, J. E., and Rosenbloom, P. S. (1995). The evolution of the Soar cognitive architecture. In D. M. Steier and T. M. Mitchell (Eds.), *Mind matters*. 1-50. Hillsdale, NJ: LEA.

3. Rosenbloom, P. S., Laird, J. E., and Newell, A. (1992). *The Soar papers: Research on integrated intelligence*. Cambridge, MA: MIT Press. Vol. 1 and 2.

4. Ritter, F. E., Baxter, G. D., Avraamides, M., and Wood, A. B. (2001). Soar Frequently Asked Questions (FAQ) list, acs.ist.psu.edu/soar-faq/

5. The Soar group web pages, ai.eecs.umich.edu/soar/soar-group.html

## Glossary

Unified theory of cognition#A theory of cognition that proposes a single set of mechanisms to explain all of cognition and behavior.

cognitive architecture#A theory proposing a set of fixed mechanisms to account for aspects of human cognition that constant across tasks, where what varies is the knowledge needed to perform the task.

expert system#A computer program designed to recreate the performance of an expert on a task.

cognitive model#A computer program designed to generate the same behavior as a human processing information in the same way.

problem space#A set of operators and a state representation that organize knowledge.

knowledge level#A level of analysis that looks at the amount of knowledge that system knows, not how it processes or uses that knowledge.

problem space level#A level of analysis of behavior that looks at behavior in terms of problem spaces and their parts.

symbol level#A level of analysis of behavior that looks at the mechanisms that give rise to behavior on the problem space level.

impasse#An architectural feature when problem solving is stopped because no knowledge can be applied (i.e., no rules match).

chunk#With respect to Soar, a rule learned when an impasse is resolved.

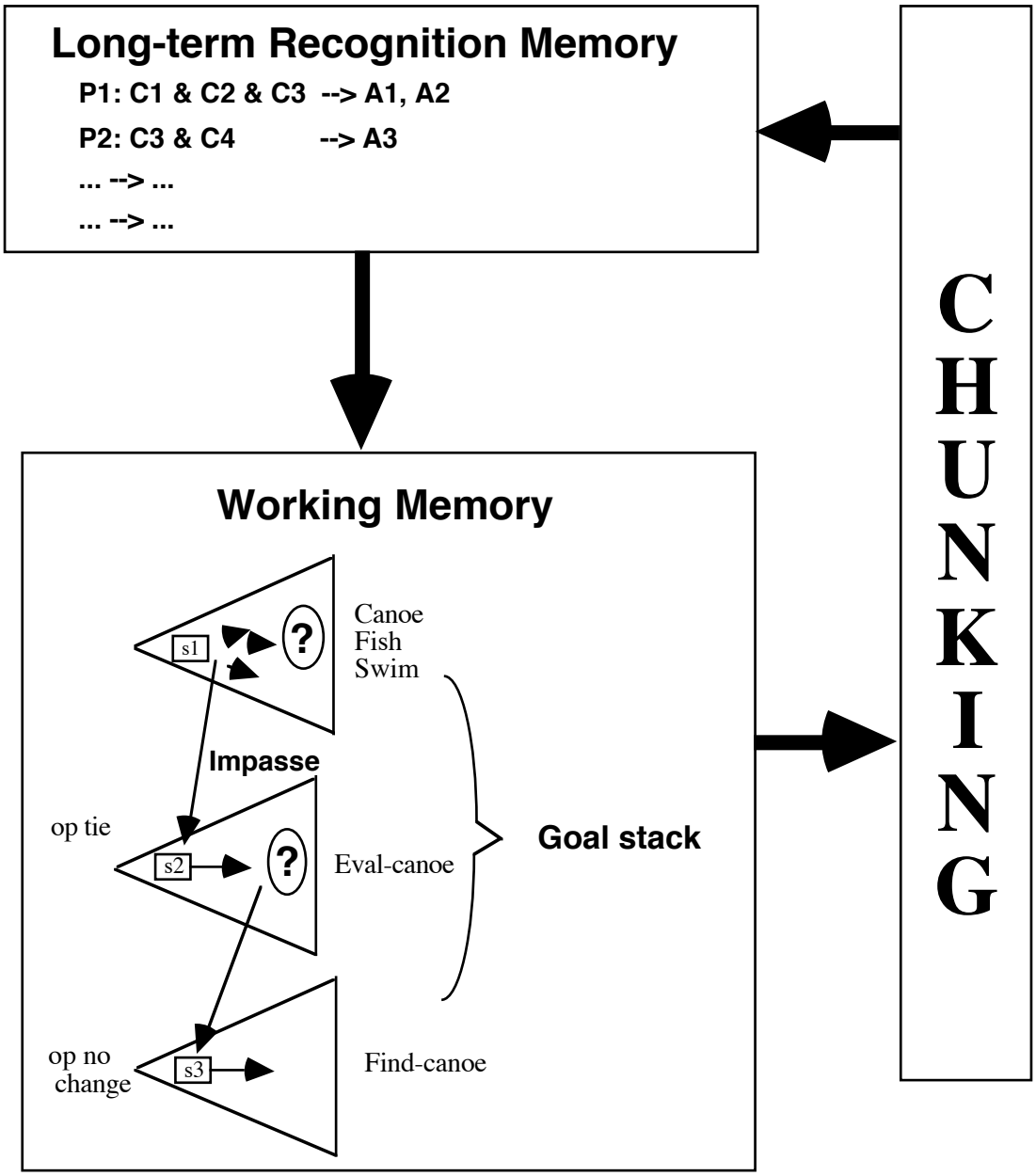


**Figure 1.** A graphic description of structures in Soar.